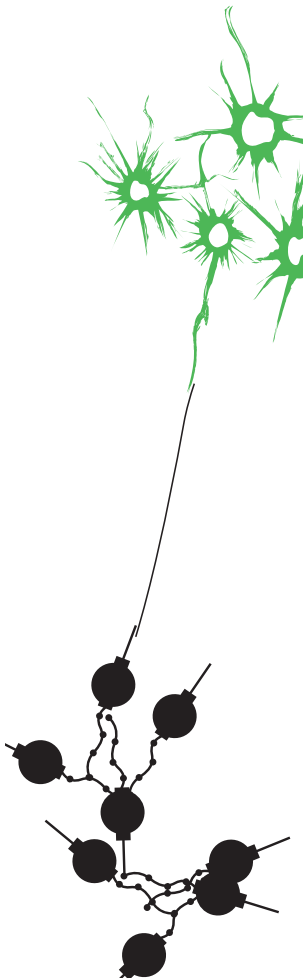


ROBOTICS AND MECHATRONICS

DEVELOPING QUADRATIC PROGRAMMING CONSTRAINTS FOR HUMAN SAFETY: AN APPLICATION OF UNIFIED SAFETY CRITERIA TO MULTI-TASK ROBOT CONTROL

M.C. (Cham) Bustraan

MSC ASSIGNMENT

Committee:
prof. dr. ir. S. Stramigioli
dr. ir. D. Dresscher
dr. ir. W. Roozing
dr. ir. J.J. de Jong

March, 2021

009RaM2021
Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE

UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Developing quadratic programming constraints for human safety: an application of unified safety criteria to multi-task robot control.

Martijn Chamroeun Bustraan

***Abstract*—As more and more applications of robots are in the vicinity of or in collaboration with humans, human safety is a topic of growing importance. The incorporation of human safety awareness into robotic manipulator control schemes has therefore become a broadly researched topic in recent years. At the same time, a lack of research on human safety in the context of balancing humanoid control methods has been identified [1]. This paper seeks to make up for this lack by proposing an implementation of human safety awareness as inequality constraints for quadratic programs, an optimization process frequently used by humanoids and other multi-tasking robots. The proposed constraints are developed based on the mechanisms of human-robot collisions and the unification of measures used to assess the severity of human injuries. The constraints are then validated using a 6-DoF manipulator in 50 randomized scenarios. The results show that the constraints improve the overall safety of a human obstacle but cannot guarantee it due to insufficient robustness of the constraints.**

I. Introduction

In the past decades, developments in the field of robotics has enabled the creation of increasingly intelligent and dexterous machines. Traditional industrial manipulators are large, heavy, and their control does not take the presence of humans into account so a physical safety barrier is constructed around them. Modern robot designs strive to incorporate human safety protocols into their control, allowing them to safely collaborate with humans.

Guaranteeing the safety of humans in close proximity to a robot is heavily researched but remains a challenging task. When no contact between a robot and human is required, one can keep the human safe by gradually slowing and eventually stopping the robot's motion as a person approaches [2], [3]. Robotic manipulators have also been successfully controlled to actively avoid obstacles when planning their paths, thereby improving productivity while maintaining human safety [4], [5]. When the collaboration between human and robot does require contact during operation other methods for guaranteeing safety have to be considered. The state-of-the-art approaches for ensuring human safety during physical human-robot interaction (pHRI) monitors one or more metrics related to human injuries (e.g. contact force, pressure, velocity, energy, or power) and adapts or extends a control method to ensure these remain within safe ranges [6]–[9].

In parallel to these studies, the design and control of self-balancing humanoids is another subject that is undergoing significant development. Self-balancing humanoids are typically bipedal or use a two-wheeled base to navigate their environment. Such designs make the robot more agile and widely applicable compared to fixed-base or non-balancing robots at the cost of being more complex systems. While walking humanoids have been made for at least 20 years ([10], [11]), studies are still being performed to improve walking stability and robustness and explore new methods of control [12]–[14]. Simultaneously, other recent research in humanoid robotics includes performing multiple tasks like walking, obstacle avoidance, disturbance rejection, and interacting with the environment all at once while maintaining balance [15]–[17]. An often chosen method to make the best use of a humanoid's flexibility in accomplishing all these tasks despite their complexity is optimization-based control; quadratic programming (QP) in particular is a popular choice of optimization process [16].

Literature seeking to apply the concepts of human safety to the control scheme of a balancing humanoid is scarce. Recently, Aller et al. [1] performed a survey of over 200 publications to establish the readiness of humanoid robots for real-world applications, their performance in locomotion tasks, and bench-marking scenarios and performance indicators used. In this work they identified a lack of research on establishing criteria and bench-marking for human safety. The goal of this paper is to address this deficiency by beginning to bridge the gap between human safety concepts developed using manipulators and the multi-task control methods deployed in humanoid control.

To achieve this, the concepts of human safety awareness must be implemented within an optimization-based controller. Very few papers have been published on this topic and those that have remain limited to a single task and/or make use of optimization processes that are not commonly used in humanoid control. The work by Meguenani et.al. in [18] and [19] presents constraints for the energy of the robot and the contact force during collision which are then used in a QCQP (quadratically constrained quadratic program) optimization process. Their contact force constraint requires another controller to set a desired trajectory for the contact point which cannot be done during multi-task control, where a desired trajectory can be defined based on each task. [20] also created a contact force constraint for a 1-DoF contact model using IFT (Iterative Feedback Control) which was then imposed on a SQP (Sequential Quadratic Program). Humanoid control methods based on QCQP or SQP can be made but QP remains most used due to its theoretical and practical efficiency

[21]. However, no literature could be found on human safety constraints developed for a QP-based controller.

Hence, in this work we present QP-compatible safety constraints intended to ensure the safety of humans is maintained while in proximity to a (multi-tasking) robot. The set of constraints is made concise by considering the interrelation of the injury measures and safety criteria. Once these constraints have been determined they are evaluated in simulation using a 6-DoF manipulator.

The rest of this paper is structured as follows. In section II essential theoretical background is discussed. Based on this, safety criteria are chosen in section III which are then translated into QP-compatible constraints in section IV. The simulation setup and robot used to validate the constraints is described in section V and the control of the robot is presented in section VI. The results of the validation experiments are presented and analyzed in section VII. Lastly, conclusions and recommendations for future work are found in section VIII.

II. Background

In order to develop human safety constraints for QP-based robot control it is necessary to first cover a number of underlying topics. First, the dynamics of human-robot collisions (HRCs) and their characteristic force profile are discussed. Second, a summary of human injury measures and safety metrics is presented which is followed by a discussion on the interrelation between these. Finally, a description of the quadratic programming optimization process is given.

A. Human-robot collision dynamics

Before human safety awareness can be incorporated into any form of control it is necessary to understand the dynamics of the collision and its characteristic force profile over time. In [22], Haddadin and Croft give an overview of the current state of the art in pHRI, including how a HRC can be modeled and its force profile characterized. During a collision the equations of motion of a robot with n joints can be written as:

$$\mathrm{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathrm{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) + \boldsymbol{\tau}_D(\mathbf{q},\dot{\mathbf{q}}) = \boldsymbol{\tau} + \boldsymbol{\tau}_e \tag{1}$$

Where $\mathbf{q} \in \mathbb{R}^n$ is the vector of joint positions, $\mathrm{M}(\mathbf{q}) \in \mathbb{R}^{n\times n}$ is the joint-space inertia matrix, $\mathrm{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} \in \mathbb{R}^n$ is the torque vector due to Coriolis and centripetal forces, $\mathbf{g}(\mathbf{q}) \in \mathbb{R}^n$ is the torque vector due to gravity, $\boldsymbol{\tau}_D(\mathbf{q},\dot{\mathbf{q}}) \in \mathbb{R}^n$ is the torque vector due to joint friction, $\boldsymbol{\tau} \in \mathbb{R}^n$ are the applied actuator torques, and $\boldsymbol{\tau}_e \in \mathbb{R}^n$ are the joint torques due to external wrenches.

The twist of a contact point c on the robot with respect to and expressed in the inertial frame is denoted as $\mathbf{T}_c^{0,0}$. Since this point is on the robot its motion must also be directly related to the robot's joint positions and velocities. The twist of a frame is related to the joint velocities through:

$$\mathbf{T}_c^{0,0} = \mathrm{J}_c(\mathbf{q})\,\dot{\mathbf{q}}\,,\ \mathrm{J}_c = \begin{bmatrix}\mathrm{J}_{c,\omega}(\mathbf{q})\\ \mathrm{J}_{c,v}(\mathbf{q})\end{bmatrix} \tag{2}$$

Where $\mathrm{J}_c(\mathbf{q}) \in \mathbb{R}^{6\times n}$ is known as the geometric Jacobian for the frame at the contact point.

The external wrench applied to the robot at the contact point expressed in the inertial frame is denoted as $\mathbf{W}_e^0$. The power transferred to the robot should be the same regardless of whether twists and wrenches or joint velocities and torques are used. Hence, $\boldsymbol{\tau}_e$ can then be obtained through

$$\begin{aligned}\mathbf{W}_e^0\,\mathbf{T}_c^{0,0} &= \boldsymbol{\tau}_c^\top\,\dot{\mathbf{q}}\\ \mathbf{W}_e^0\,\mathrm{J}_c(\mathbf{q})\,\dot{\mathbf{q}} &= \boldsymbol{\tau}_e^\top\,\dot{\mathbf{q}}\\ \boldsymbol{\tau}_e &= \mathrm{J}_c(\mathbf{q})^\top\,\mathbf{W}_e^{0,c^\top}\end{aligned} \tag{3}$$

Using the Jacobian of the contact point it is also possible to determine the effective mass m_c of the robot acting in the collision direction:

$$m_c(\mathbf{q}) = \left[\hat{\mathbf{u}}^\top(\mathbf{q})\,\mathrm{J}_{c,v}(\mathbf{q})\,\mathrm{M}^{-1}(\mathbf{q})\,\mathrm{J}_{c,v}^\top(\mathbf{q})\,\hat{\mathbf{u}}(\mathbf{q})\right]^{-1} \tag{4}$$

Where $\hat{\mathbf{u}}(\mathbf{q}) \in \mathbb{R}^3$ is a unit vector in the collision direction from the contact point.

***Collision force profile**:* Typically, the force applied during a collision between a robot and human consists of two distinct consecutive phases. First there is a short impact phase which is followed by a quasi-static contact phase [22].

The impact phase can be well understood using the modeling done above. During this initial impact period the entirety of the robot can be condensed to a point mass with mass m_c, velocity $\mathbf{v}_c^{0,0}$, and a description of its curvature (the effect of which goes beyond the scope of this work). The characteristics of the colliding body part also play an important role, especially its reflected inertia and contact stiffness. This phase can then be well understood using a relatively simple mass-spring-mass model. The force magnitude profile of such a model contains a single peak the magnitude of which computed as [22]:

$$F_{imp}^{\max} = \sqrt{\frac{m_c\,m_h}{m_c+m_h}}\sqrt{k_h}\,v_{c,0}^h \tag{5}$$

Where $F_{imp}^{\max}$ is the maximum contact force due to this phase, m_c is the effective mass of the robot as defined in (4), m_h is the reflected inertia of the human, k_h the human contact stiffness, and $v_{c,0}^h$ the relative initial velocity between the robot and human.

During the quasi-static contact phase the robot applies either a pushing force when the struck body part is free to move or a crushing force when it is clamped into place. This phase is highly dependent on the robot's design and control and is especially important when the human is clamped since then the largest forces are applied. The force applied by the robot at the point of contact can be found through

$$\mathbf{F}_c = \mathrm{J}_{c,v}^{\top\dagger}(\mathbf{q})\,\boldsymbol{\tau} \tag{6}$$

Where $\mathrm{J}_{c,v}^{\top\dagger}(\mathbf{q})$ is the pseudoinverse of the transposed geometric jacobian of the contact point. Naturally, the contact force could be large enough to penetrate or break human tissue/structure and so it is also limited by the human tissue resistance to crushing.

For free collisions, the maximum applied force in this second phase is typically lower than that of the first for

robot velocities greater than $0.3\,\mathrm{m\,s^{-1}}$ [22]. For free collisions against soft-tissue, these two phases could even simplify into a single initial impact phase after which contact is lost again. If the pushing force during this phase does the exceed peak force of the first phase it is predominantly driven by the actuator torques and the reaction of the human body which is mainly governed by its reflected impedance. However, the applied pushing force could still never exceed the crushing force that would be applied if the human was clamped, with all other things being equal.

B. Injury measures and safety metrics

The human body can sustain many types of injuries including bone fractures, internal injuries, abrasions, and lacerations due to being cut, stabbed, gashed, or crushed. In [22] Haddadin and Croft identify under what conditions each of these forms of injury is likely to occur and what factors determine its severity. Tadele et. al. in [23] have performed a literature review on current safety metrics used to embed safety-awareness into robots. Both their findings are summarized here to gain insight into what causes injuries and what metrics are used to mitigate them.

***Contact force**:* According to [22] the force applied during contact is an effective injury measure for each form of injury and contact. [23] has also found works that consider large forces as the cause of potential injuries and thus implement force-based criteria into control. The human body's tolerance to contact forces varies depending on the struck part, with the neck having the lowest tolerance [23]. Some control the applied force directly to implement safety, others have defined new safety metrics based on force such as the danger index or impact potential [23].

***Compression**:* Haddadin and Croft in [22] state that a measure known as the compression criterion can act as an injury measure for fractures as well as crushing and gashing lacerations during blunt contacts. The compression criterion is the compression of the chest expressed as a percentage of the original chest depth [24]. [23] discusses a similar injury measure called the viscous criterion as an injury measure for constrained organ damage. The viscous criterion is defined as the product of the rate of compression and the percentage of compression [24]. The rate of kinetic energy transfer has been suggested as a viscous criterion safety metric [23]. It is important to note that both the compression criterion and viscous criterion are by definition only relevant during collisions against the human chest and abdominal areas.

***Energy**:* Tadele et. al. in [23] state some consider a large amount of uncontrolled energy is a cause of accidents in robots and have identified energy limits that cause fractures of the skull and damage the spinal cord. In contrast to this, [22] suggests that energy is an effective energy measure only in case of dynamic loading of a sharp contact which can cause cutting and stabbing lacerations and abrasions. [22] chooses to relate energy to fractures in different manner as will be discussed later. Many works have effectively implemented safety-awareness into their control methods through controlling the energy of the robot, see for instance [8], [18], [19] or [25].

***Energy density**:* Instead of energy, [22] states that energy density acts as an injury measure for fractures and other injuries during the dynamic loading of a blunt contact. Both [22] and [23] state a strong correlation exists between the pain felt by a human and the impact energy density. [23] goes on to state that energy density limits of the skin have been used to design safer robot coverings. [22] does not explicitly provide its reasoning for using energy density instead of energy for blunt contacts beyond that its used to evaluate lacerations. It is thought that the authors of [22] considered force and compression sufficient to adequately measure fracture injuries and density was added to evaluate possible lacerations during the contact.

***Stress**:* Stress is indicated as an injury measure for all forms of sharp contact by [22]. The review by Tadele et. al. [23] agrees with this as they describe the use of stress as a safety metric for mitigating skin injuries and soft-tissue injuries, which can be caused by sharp contacts. The works referenced by [23] all use stress (some in conjunction with energy density) in order to design the shape, elastic modulus, and thickness of a robot covering.

***Others**:* Haddadin and Croft in [22] also discuss injury evaluation by a medical expert via the AO-classification as being the most precise method of judging the extent of any injury. This can only done after the injury has been sustained and so is used in experimental studies to define what is known as a risk curve. [6], a work also described by [23], defines such a curve where the observed injury was directly related to impactor mass, velocity, and geometry.

Acceleration based metrics are also discussed in [23] but are not included in [22] as an injury measure. Acceleration based metrics stem from the head injury criteria (HIC), used in biomechanics studies and accident research in fields such as the automotive industry. In robotics it has been used as a severity indicator for potential injury due to blunt impact to the human head [23]. However, while HIC is a standardized metric in automotive crash tests, there is a need for other safety metrics in robotics due to the difference in injury types and operating speeds [23].

C. Interrelation of safety metrics

The detailed reviews by Haddadin and Croft in [22] and Tadele et. al. in [23] show that many factors come into play when determining effective ways to measure the severity of injuries as well as robot control and design methods for mitigating them. Many of these factors are closely related to one another which raises the question if one of these factors can be carefully controlled such that all factors remain within safe ranges. Let us consider the connections between the energy of a robot and the other identified injury measures.

***Force**:* Limiting the robot's kinetic energy involved in a collision is done through controlling the component of the robot's velocity toward the human and/or the effective robot mass. These also directly relate to the maximum force applied

during the initial impact phase of a collision through (5). The applied force can also be determined by monitoring the rate of change of momentum, which makes use of the same properties as energy. The force applied during the quasi-static contact phase is highly dependent on the control method and therefore the potential energy stored in the controller. Whether the potential energy stored in the controller can be estimated depends on the type of robot used and how it is controlled. If it can be estimated then contact forces during quasi-static contact can be limited by controlling this potential energy.

***Compression**:* The compression criterion is a measure how much the chest has been compressed relative to normal. Compression of the chest is caused by the application of force and so is not directly related to energy but rather indirectly through the forces applied by the robot due its kinetic and potential energy. The viscous criterion, on the other hand, is the product of the rate of compression and the relative chest compression and therefore more directly related to energy. The rate of compression is directly related to the velocity of the robot at the contact point which can also be controlled by ensuring a low kinetic energy. Furthermore, [23] has suggested that the rate of kinetic energy transfer can act as safety metric for the viscous criterion, further suggesting that the viscous criterion can be brought to safer ranges by limiting the robot's kinetic energy.

***Energy density and stress**:* If the contact area of the collision can be determined then the energy density can also be controlled through controlling energy. Stress is a measure of the amount of force experienced by a material per unit area and so can only be related to energy through the relations between energy and force.

In order to confirm whether safety is guaranteed from injuries correlated to measures other than energy additional properties of the collision must be monitored. The collision time would have to be known in order to determine the rate of energy transfer during the collision which can be used as a metric for the viscous criterion. The compression distance should be monitored to gain insight into the compression criterion and viscous criterion. If the potential energy of the controller and the applied force due to it cannot be determined then another method for constraining this force would have to be found. As mentioned above, the contact area of the collision needs to be known to determine the danger due to energy density and stress.

In this way one can see how by controlling energy other injury measures can also be brought to safer ranges. The effective mass and/or velocity of the robot in a collision influences every injury measure except applied force due to controller potential energy. Kinetic energy is the only injury measure that is directly dependent on both of these and so can be considered a central injury measure that is easily related to all the other measures. Even the applied force, which is a relevant measure for all types of injury and contact, can be considered less central when realizing that the applied force in the first phase of contact is turn dependent on the properties that make up the robot's kinetic energy as shown in (5). Whether the control of a single injury measure such as energy can be proven to yield sufficient safety from all forms of injury in a general form is an interesting and useful question to answer, but one which goes beyond the scope of this work.

D. Quadratic programming

Practically all humanoids have a large number of degrees of freedom in branched structure. This gives them a high degree of redundancy and thus many possible ways to complete its goal(s). Modern methods of humanoid control maximize the humanoid's agility and versatility in multi-task situations through whole-body control [12].

A whole-body control method should be capable of achieving multiple objectives such as remaining in balance while completing tasks with its hands. The humanoid's motions should be able to conform to certain constraints like preventing self-collisions or maintaining certain contact forces. Analytically finding a solution for a vector of inputs that achieves all goals and conforms to all constraints is a challenging task. It is made more complex by the fact that a range of solutions could be possible and tasks and constraints can be added and removed depending on the current environment and user wishes. As, such most if not all humanoid whole-body control frameworks make use of optimization processes.

A popular type of optimization process in humanoid control is known as quadratic programming due to the availability of solvers that are both fast and reliable in conjunction with the availability of more powerful CPU's [16]. Mathematically it can be phrased as:

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^\top \mathrm{Q}\,\mathbf{x} + \mathbf{f}^\top \mathbf{x} \tag{7}$$

$$\text{s.t.} \begin{cases} \mathrm{A}\,\mathbf{x} \leq \mathbf{b} \\ \mathrm{A}_{eq}\,\mathbf{x} = \mathbf{b}_{eq} \\ \mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{x}_{\max} \end{cases} \tag{8}$$

The goal of QP is to find the vector of arbitrary length $\mathbf{x} \in \mathbb{R}^k$ that minimizes the objective function given in (7) while conforming to the constraints stated in (8). In order for the optimization to work $\mathrm{Q} \in \mathbb{R}^{k\times k}$ must be a symmetric real matrix and $\mathbf{f} \in \mathbb{R}^k$ a real vector. Any linear combination of the elements of $\mathbf{x}$ can be set as a constraint through $\mathrm{A} \in \mathbb{R}^{j\times k}$ and $\mathbf{b} \in \mathbb{R}^j$ or through $\mathrm{A}_{eq} \in \mathbb{R}^{j\times k}$ and $\mathbf{b}_{eq} \in \mathbb{R}^j$ for inequality and equality constraints, respectively; where j indicates the number of constraints. Finally, the output can be constrained to remain within a designated range through $\mathbf{x}_{\min} \in \mathbb{R}^k$ and $\mathbf{x}_{\max} \in \mathbb{R}^k$.

In humanoid control literature QP is used to minimize a large variety of different functions depending on the context in which QP is used. The work in [12] is concerned with whole-body balancing in multi-contact scenarios, where a QP optimization is used for the distribution of the desired CoM wrench among the end-effectors in contact with the environment. QP is used in [14] in the context of standing-up control to ensure the control commands follow the laws of contact-consistent operational space control and contact conditions related to the center of pressure and friction cones

simultaneously. [16] makes use of whole-body momentum in combination with finding joint accelerations and ground reaction forces in a single QP formulation. The framework also includes a method to specify various motion tasks which are also taken into account by the optimization process. In [17] a QP-based low-level controller is used to generate body torques based on operational space objectives and center of mass (CoM) motion policies provided by a high-level controller. [26] make of use of an unconstrained QP formulation to optimize the realization of instantaneous capture point position goals, base frame velocity goals, and torque input minimization during a 2D push recovery of a wheeled-base humanoid.

Most, if not all, operate by choosing (7) to be a error signal between desired and actual properties of the humanoid. This is typically expressed as a function dependent on, among others, variables that drive the robot such as its joint accelerations or torques.

III. Establishing safety criteria

In this section controllable safety criteria will be proposed for guaranteeing a human's safety within a robot's workspace. The collision between a human and robot is modeled as described in section II-A. First the assumptions made for the modeling of the obstacles and contact are discussed. Then safety criteria are proposed for both phases of contact discussed in section II-A.

A. Assumptions

Throughout the rest of this work the following assumptions are made with respect to representing human obstacles and modeling the contact between a human obstacle and the robot:

- The positions of all nearby obstacles is known to the robot at all times.
- The human obstacle is considered to be a single point, representing the center of a human body part, with a sphere around it. Anything inside this sphere is considered in contact with the human obstacle.
- Contact is considered to be a one-dimensional collision along the direction of contact and the effects due to the collision contact area and impactor shape are neglected.
- The human obstacles cannot move and any collision is considered to clamp the obstacle.
- The dynamics of a collision between the human obstacle and the robot are independent of the direction and angle of impact.

The detection of obstacles, human or otherwise, is an ongoing topic of research and essential for enabling robots to accomplish their goals in an arbitrary environment. In fact, many robots already make use of advanced computer vision for obstacle detection such as Spot and Atlas developed by Boston Dynamics[1], the CRX line by Fanuc[2], and EVE by Halodi[3]. There is also high demand for advanced obstacle detection and human identification in other rapidly developing industries such as self-driving cars. Hence the assumption that a collaborative robot is equipped with similar vision capabilities is considered reasonable.

In practice a human obstacle would never be just a single point but rather a series of connected bodies not unlike a robot. Then, just as the entire robot is made equivalent to a point mass for the purposes of collision analysis, so to can the human body be made equivalent to a reflected inertia and contact stiffness experienced by this robot equivalent point mass. Of course this is still a simplification since in reality a HRC is never just between two points but between two three-dimensional bodies.

The assumption that the collision is considered one-dimensional simplifies the collision so that shear forces applied between the human and robot do not need to be modeled. The contact area and impactor shape could be taken into account even when simplifying the collision in this way, however, it was chosen to neglect these in this work in order to focus on the development of the criteria and constraints rather than the modeling of a complex collision. Due to these assumptions, the injury measures of energy density and stress cannot be taken into account.

Considering an obstacle to be clamped when in contact puts it in the most dangerous form of contact since then the largest forces are applied during the quasi-static contact phase. When a human is clamped it is considered to have an infinite reflected inertia [7]. Eliminating the motion of the obstacle entirely means the velocity of the obstacle does not play a role in the collision. While the effects of a human's motion on their safety should eventually be taken into account by a safety constraint, this was left outside of the scope of this work.

The direction and angle of impact would in reality also affect the collision. It can influence the contact stiffness and reflected inertia of the human experienced by the robot as well as the criteria limits for safe behavior. However, by making the proposed safety criteria either independent with respect to these or configurable the criteria can be extended to include direction and angle of impact in a later work.

B. Impact phase safety criteria

Following section II-A, this first phase is dominated by the impact collision properties of both bodies: the effective mass and velocity of the robot as well as the contact stiffness, velocity, and reflected inertia of the human. It can be modeled as a mass-spring-mass system where one mass has the robot's effective mass and velocity, the other the human's reflected inertia and velocity, and the spring is the contact stiffness of the human body at the point of impact.

Conform to the assumptions made, injuries during this phase can be caused by large impact forces, too much energy involved in the collision, and exceeding compression and viscous criteria limits in case of collisions against the torso. It could be possible to define a safety criteria for each of these, but by making use of the reasoning in section II-C it can be

[1]https://www.bostondynamics.com/
[2]https://crx.fanucamerica.com/
[3]https://www.halodi.com/ever3

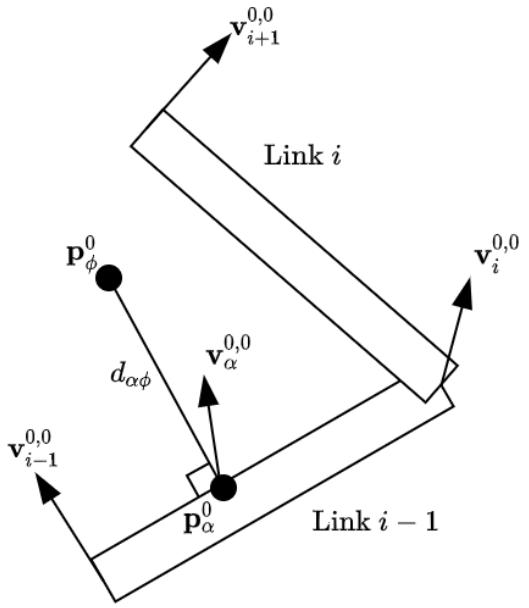


Fig. 1: Diagram illustrating the definition of the approaching point $\mathbf{p}_\alpha$ on the robot for a particular obstacle $\mathbf{p}_\phi$.

argued that an energy-based criteria alone could be sufficient to guarantee safety during this phase of the collision.

The robot has both kinetic energy due to its motion and potential energy stored in its controller and due to gravity (which is assumed to be adequately compensated). This first phase of a collision is dominated by collision dynamics and these dynamics can in turn be considered a function of the kinetic energy present during the collision. Potential energy stored in the controller can still influence this phase but only when this is converted to kinetic energy before the collision occurs. Hence, for this phase, a safety criteria will be proposed based on kinetic energy. In principle, kinetic energy involved in the collision can be supplied by either the robot or human. However, under the assumption that the human obstacle is immobile all kinetic energy must be provided by the robot.

In order to use this as a safety metric for robotic humanoids it is desirable that the kinetic energy involved in the collision is controlled while imposing minimal constraints on its motion possibilities. The robot can then still accomplish its tasks as best as possible while guaranteeing safety.

In section II-A it is discussed how the collision is modeled by reducing the robot down to an effective point mass and velocity. In order to make this reduction a point on the robot must be defined at which the effective mass and velocity is calculated. To accomplish this a point on the robot is defined called the approaching point for a particular obstacle. For a particular obstacle the approaching point on the robot is defined as the point that is closest to the obstacle and has a positive velocity toward it.

Figure 1 shows an illustration to further clarify what the approaching point is. In the figure two links of an arbitrary robot are shown along with the Cartesian velocities of their end-points relative to the inertial frame and expressed in the inertial frame. The position of the obstacle in the inertial is given by $\mathbf{p}_\phi^0$ and the position of the approaching in the inertial frame by $\mathbf{p}_\alpha^0$. While link i is closer to the obstacle than link $i-1$ it is moving away from the obstacle and therefore not suitable for the approaching point. One end of link $i-1$ is moving toward the obstacle while the other is moving away. Therefore, the approaching point is selected as the closest point to the obstacle within the segment of link $i-1$ that has a positive velocity toward the obstacle. The distance between the approaching point and the obstacle is then given as $d_{\alpha\phi}$. If no point on the robot has a positive velocity toward the obstacle then it is considered safe.

By only considering the safety of human obstacle with respect to the approaching point the robot is given as much freedom of movement as possible while still ensuring that collisions between the obstacle and the robot occur safely. The kinetic energy of the robot involved in a collision between the obstacle and the approaching point is:

$$E_{imp}(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2} m_{\alpha\phi}(\mathbf{q}) v_{\alpha\phi}(\mathbf{q}, \dot{\mathbf{q}})^2 \tag{9}$$

Where $m_{\alpha\phi}$ is the equivalent mass of the robot computed at $\mathbf{p}_\alpha^0$ in the direction of $\mathbf{p}_\phi^0$ and $v_{\alpha\phi}$ the scalar velocity of α in the direction of ϕ. Based on the information provided in section II-A, each is computed as follows:

$$m_{\alpha\phi}(\mathbf{q}) = \left(\mathbf{J}_{\alpha\phi}(\mathbf{q})\, \mathrm{M}^{-1}(\mathbf{q})\, \mathbf{J}_{\alpha\phi}^\top(\mathbf{q})\right)^{-1} \tag{10}$$

$$v_{\alpha\phi}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{J}_{\alpha\phi}(\mathbf{q})\, \dot{\mathbf{q}} \tag{11}$$

$$\mathbf{J}_{\alpha\phi}(\mathbf{q}) \equiv \hat{\mathbf{u}}_{\alpha\phi}^\top(\mathbf{q})\, \mathrm{J}_{\alpha,v}(\mathbf{q}) \tag{12}$$

Where $\mathrm{M}(\mathbf{q})$ is the joint-space mass matrix of the robot, $\mathrm{J}_{\alpha,v}(\mathbf{q})$ the Jacobian relating the joint velocities, $\dot{\mathbf{q}}$, to the Cartesian velocity of the approaching point $\mathbf{v}_\alpha^{0,0}$, and $\hat{\mathbf{u}}_{\alpha\phi}(\mathbf{q})$ the unit vector pointing from $\mathbf{p}_\alpha^0$ to $\mathbf{p}_\phi^0$. Note that $v_{\alpha\phi}$ must by definition be positive since $\mathbf{p}_\alpha^0$ must be approaching the obstacle.

The initial impact phase of a collision is considered safe as long as

$$E_{imp}(\mathbf{q}, \dot{\mathbf{q}}) \leq E_{lim} \tag{13}$$

holds when the collision occurs. Activating this constraint just before a collision occurs could force to robot to have to try to reduce its impact energy by a large amount in a small time window. This could be physically impossible, cause erratic behavior, or impair essential functions such as gravity compensation or balancing. At the same time, forcing the robot to operate with low energies while the obstacle is relatively far away can be considered an unnecessary limit on performance. To balance these, the energy limit of the approaching point is defined as a function of the distance separating it from the obstacle:

$$E_{lim}(d_{\alpha\phi}) = \begin{cases} E_{safe} & d_{\alpha\phi} \leq d_{safe} \\ E_{safe} + \kappa_E (d_{\alpha\phi} - d_{safe}) & d_{\alpha\phi} > d_{safe} \end{cases} \tag{14}$$

Where d_{safe} is the maximum distance for which the kinetic energy is limited to E_{safe}. For distances larger than d_{safe}, the constraint is linearly relaxed at a rate of κ_E.

Naturally, the value of E_{safe} will depend on what degree of harm is considered unsafe, where on the human body the collision occurs, and whether the human is clamped. For instance, in [22] a number of experiments were performed wherein impacts were made on the lateral surface of the upper arm while it was clamped. The pain felt by the target was recorded using the visual analog scale (VAS) where a score of 0 indicates no pain and a score of 10 the worst pain possible. If the initial impact of a collision is deemed safe as long as it results in little to no pain to the upper arm, a score of 2 or lower, then E_{safe} should be set to $4.5\,\mathrm{J}$.

Energy can be controlled through either the effective mass or effective velocity of the approaching point toward the obstacle. The velocity of the robot is more straightforward to control and has a squared relationship with the kinetic impact energy. Keeping in mind that the velocity of the approaching point must be greater than zero, the impact energy criteria can be rewritten as:

$$v_{\alpha\phi}(\mathbf{q}, \dot{\mathbf{q}}) \leq \sqrt{\frac{2E_{lim}(d_{\alpha\phi})}{m_{\alpha\phi}(\mathbf{q})}} \tag{15}$$

C. Quasi-static contact phase safety criteria

After the initial impact of the collision has taken place, the robot could keep applying a force on the human in its attempt to complete a task. During this phase of the collision in combination with the assumption that the human is clamped, the robot applies a crushing to the human obstacle. The contact force is listed by [22] as a useful injury measure in each possible collision type. Therefore, ensuring the contact force remains safe could in theory guarantee safety from all injuries during this phase of the collision.

These contact forces can be thought of as arising from potential energy stored in the controller that is seeking to push the contact point further toward the human. For manipulators, this potential energy of the controller can be estimated in various ways as shown in [8] and [19]. When considering humanoids, these same concepts become difficult to apply since it is not driven by a single control goal, but continually seeking to optimally achieve multiple goals at once. Since the potential energy stored in the controller is not guaranteed to be computable for multi-tasking robots an energy-based criteria to limit the contact forces in this phase cannot be employed.

Instead, we propose to consider criteria for quasi-static contact safety only once contact has been established. The detection, localization, and estimation of external contact forces is possible (e.g. [27] and [28]) and can be used as the trigger for when this criteria should be monitored.

The obstacle is considered as a single point, however, any part of the robot within a sphere surrounding it is considered in contact with the obstacle. For each link with a segment within this sphere, its contact point is defined as the point on the link that is closest to the obstacle.

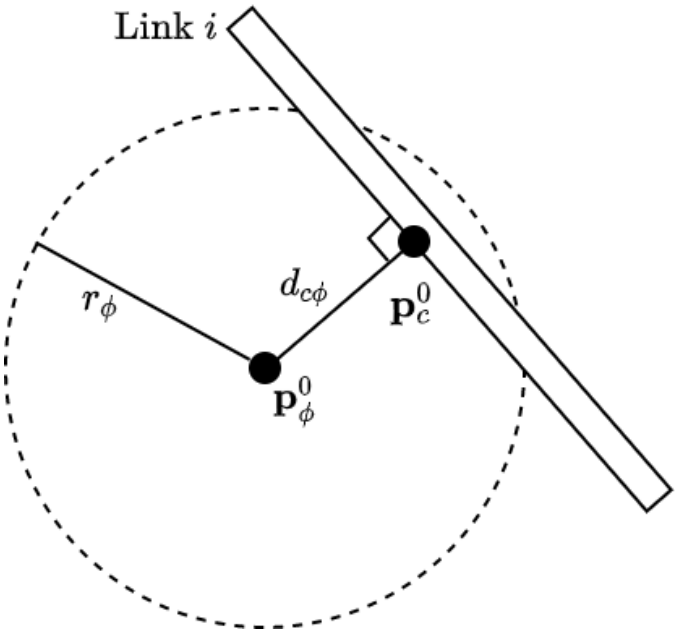


Fig. 2: Diagram illustrating the definition of a contact point $\mathbf{p}_c$ on the robot for a particular obstacle $\mathbf{p}_\phi$.

The definition of a contact point on a link is illustrated in figure 2. The obstacle is positioned with respect to the inertial frame at $\mathbf{p}_\phi^0$ and has a contact radius of r_ϕ. A segment of an arbitrary link is within this contact radius and so the contact point on the robot, $\mathbf{p}_c^0$, is defined as the point on the link closest to the obstacle. The distance between $\mathbf{p}_c^0$ and $\mathbf{p}_\phi^0$ is then given by $d_{c\phi}$. Note that if two links are within the contact radius then a contact point is defined for each.

The effective force applied by the robot at a contact point can be determined the same way as shown in II-A, namely:

$$\mathbf{F}_c = \mathrm{J}_{c,v}^{\top\dagger}(\mathbf{q})\,\boldsymbol{\tau} \tag{16}$$

The component of this force that is directed toward the human is then isolated as follows:

$$F_{c\phi} = \hat{\mathbf{u}}_{c\phi}^{\top}(\mathbf{q})\,\mathbf{F}_c \tag{17}$$

Where $\hat{\mathbf{u}}_{c\phi}(\mathbf{q})$ is unit vector pointing from $\mathbf{p}_c^0$ to $\mathbf{p}_\phi^0$. A human in quasi-static contact with a robot should be safe from injuries due to the contact force so long as

$$F_{c\phi} \leq F_{safe} \tag{18}$$

for the duration of the contact period. The value of F_{safe} is of course dependent on which part of the human is at risk of being crushed. In [7], Lucci et. al. state that for a clamped upper arm the applied force should not exceed $150\,\mathrm{N}$.

IV. Deriving safety constraints

In the preceding section two safety criteria were determined, the first to ensure safety during the initial impact phase of a collision and a second for the quasi-static contact phase that follows. In this section these criteria will used to formulate

constraints that are compatible with a quadratic program. As discussed in section II-D, many humanoid robots use QP optimization to determine the optimal control signal for achieving its tasks as best as possible while adhering to a set of constraints. Human safety is perhaps the most important thing a robot should able to guarantee in all its actions. The control signals found by the QP solver must adhere to the constraints imposed on it. Hence, the safety criteria described in section III will be formulated as inequality constraints for a QP solver.

Depending on the goals a multi-tasking robot is attempting to achieve the variables for which the QP is minimized can vary but the joint torques and/or joint accelerations are practically always included. In order to formulate constraints that can be implemented with little or no changes into any humanoid control scheme, we will show that the developed criteria can be expressed as either joint acceleration or torque constraints.

To achieve this, we use a discrete linear approximation of the robot's equations of motion. Assuming that $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$ are known at discrete time k, we approximate $\mathbf{q}$ and $\dot{\mathbf{q}}$ at time $k+1$ as:

$$\dot{\mathbf{q}}_{|k+1} = \dot{\mathbf{q}}_{|k} + \delta t \ddot{\mathbf{q}}_{|k} \tag{19}$$

$$\mathbf{q}_{|k+1} = \mathbf{q}_{|k} + \delta t \dot{\mathbf{q}}_{|k} + \frac{\delta t^2}{2} \ddot{\mathbf{q}}_{|k} \tag{20}$$

Where δt is the time-step size between times k and $k+1$. The joint acceleration applied over this time-step, $\ddot{\mathbf{q}}_{|k}$, is assumed to be constant. It is either obtained from the QP solver directly or, if the solver returns joint torques, can be calculated through rewriting equation (1) into:

$$\ddot{\mathbf{q}}_{|k} = \mathrm{M}^{-1}(\mathbf{q}_{|k}) \left(\boldsymbol{\tau}_{|k} + \boldsymbol{\tau}_{e|k} - \boldsymbol{\gamma}_{|k} \right) \tag{21}$$

With

$$\boldsymbol{\gamma}_{|k} \equiv \mathrm{C}(\mathbf{q}_{|k}, \dot{\mathbf{q}}_{|k})\, \dot{\mathbf{q}}_{|k} + \mathbf{g}(\mathbf{q}_{|k}) + \boldsymbol{\tau}_{D|k}(\mathbf{q}_{|k}, \dot{\mathbf{q}}_{|k}) \tag{22}$$

Where the assumption is made that the applied actuator torque over the time-step, $\boldsymbol{\tau}_{|k}$, is constant.

A. Initial impact constraint

Given that $\mathbf{q}_{|k}$ and $\dot{\mathbf{q}}_{|k}$ are known at time k, we would like to formulate a constraint for $\ddot{\mathbf{q}}_{|k}$ such that $E_{imp|k+1}$ is less than or equal to $E_{lim|k+1}$. By substituting (19) and (20) into (9) the impact energy at time $k+1$ as a function of $\ddot{\mathbf{q}}_{|k}$ can be determined. However, this would be a non-linear function with respect to $\ddot{\mathbf{q}}_{|k}$ and, as shown in (8), inequality constraints for a QP must be linear with respect to the input vector.

Therefore, $\mathbf{q}_{|k}$ and $\dot{\mathbf{q}}_{|k}$ will be used to determine the constraint instead. Since both of these are known $\mathbf{J}_{\alpha\phi}$, M, $\boldsymbol{\gamma}$, $\boldsymbol{\tau}_e$, and $d_{\alpha\phi}$ can then be computed. However, in order for the constraint to then effectively limit $E_{imp|k+1}$ it must be assumed that $\mathbf{q}$, $\dot{\mathbf{q}}$ and the parameters dependent on them do not change significantly over time. This does introduce error into the constraint, the significance of which should be evaluated.

Starting from (15), a linear constraint for the joint accelerations imposed by an obstacle can then be derived as:

$$\begin{aligned} v_{\alpha\phi|k+1} &\leq \sqrt{\frac{2E_{lim}}{m_{\alpha\phi}}} \\ \mathbf{J}_{\alpha\phi}\, \dot{\mathbf{q}}_{|k+1} &\leq \sqrt{\frac{2E_{lim}}{m_{\alpha\phi}}} \\ \delta t \mathbf{J}_{\alpha\phi}\, \ddot{\mathbf{q}}_{|k} &\leq \sqrt{\frac{2E_{lim}}{m_{\alpha\phi}}} - \mathbf{J}_{\alpha\phi}\, \dot{\mathbf{q}}_{|k} \end{aligned} \tag{23}$$

Substituting (21) for $\ddot{\mathbf{q}}_{|k}$ then yields an equivalent constraint for the joint torques:

$$\delta t \mathbf{J}_{\alpha\phi}\, \mathrm{M}^{-1}\, \boldsymbol{\tau}_{|k} \leq \sqrt{\frac{2E_{lim}}{m_{\alpha\phi}}} - \mathbf{J}_{\alpha\phi}(\dot{\mathbf{q}}_{|k} + \delta t \mathrm{M}^{-1} (\boldsymbol{\tau}_e - \boldsymbol{\gamma})) \tag{24}$$

B. Quasi-static contact constraint

The criteria to guarantee crushing forces remain safe can also be phrased as a linear inequality constraint. However, just as with the impact constraint, $\hat{\mathbf{u}}_{c\phi}$, $\mathrm{J}_{c,v}^{\top\dagger}$, M, $\boldsymbol{\gamma}$ and $\boldsymbol{\tau}_e$ must be computed using $\mathbf{q}_{|k}$ and make use of the assumption that they do not change significantly over a single time-step. The significance of the error introduced by this assumption must also be evaluated. The constraint for joint torques can now be obtained in a straightforward way by using (16), (17), and (18), namely:

$$\mathbf{J}_{c\phi,F}\, \boldsymbol{\tau}_{|k} \leq F_{safe} \tag{25}$$

Where $\mathbf{J}_{c\phi,F} \equiv \hat{\mathbf{u}}_{c\phi}^{\top} \mathrm{J}_{c,v}^{\top\dagger}$ Using (21) we can also derive a constraint in terms of joint accelerations:

$$\mathbf{J}_{c\phi,F}\, \mathrm{M}\, \ddot{\mathbf{q}}_{|k} \leq F_{safe} + \mathbf{J}_{c\phi,F}\, (\boldsymbol{\tau}_e - \boldsymbol{\gamma}) \tag{26}$$

V. Simulation Setup

In this section the simulation setup used for validating the developed safety constraints is discussed. First a kinematic model of the used robot is described, followed by establishing its dynamic properties. Finally, the implementation of contact forces applied by the human on the robot is presented.

Applying the QP constraints to a humanoid robot was not possible as those available did not provide access to their QP solver. Instead of applying the constraints through the humanoid control API, it was chosen to design a manipulator in MATLAB in order to have full access to the QP solver. While this manipulator does not have a multi-task control framework it does enable testing of the QP-compatible constraints proposed in section IV.

A. Kinematic description

A schematic 2D diagram of the used robot viewed in the yz-plane is shown in figure 3. It is a serial manipulator consisting of six links and has six degrees of freedom. The frame Ψ_0 shows the position of the world frame. The frames Ψ_1 through Ψ_6 are each placed at the base of a link and oriented such

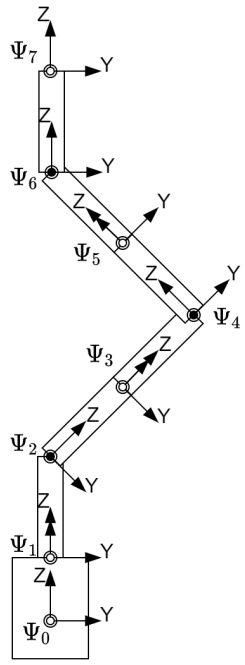


Fig. 3: Schematic diagram of the robot viewed in the yz-plane. A revolute joint at a frame's origin is indicated with a double arrow head or a filled dot at the origin, representing rotations along the z- and x-axes respectively.

that the frame's z-axis runs along the link's length. At each of these frames there is also a joint that allows for rotation relative to the previous frame in the chain. For example, Ψ_3 (link 3) can only rotate about its z-axis relative of Ψ_2 (link 2) and Ψ_4 (link 4) can only rotate about its x-axis relative to Ψ_3. The manipulator's end-effector is at the end of the sixth link and represented by Ψ_7. In the reference configuration the manipulator is completely straight along the world z-axis with no relative rotation between any frames. The configuration shown in figure 3 then corresponds to:

$$\mathbf{q} = \begin{bmatrix} 0 & -\pi/4 & 0 & \pi/2 & 0 & -\pi/4 \end{bmatrix}^\top$$

Points on a link between the frames are parametrized as in [4]:

$$\mathbf{p}_{i,s} = \mathbf{p}_i + s(\mathbf{p}_{i+1} - \mathbf{p}_i),\ \mathbf{v}_{i,s} = \mathbf{v}_i + s(\mathbf{v}_{i+1} - \mathbf{v}_i) \quad (27)$$

Where $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ are the positions of Ψ_i and Ψ_{i+1} in the world frame while $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$ are their corresponding linear velocities, and $s \in [0,\, 1]$. This means that an infinitely thin wire model of the manipulator is used to determine the location of the approaching point and contact points on the manipulator. This simplifies the process of finding these points and their geometric Jacobians within this simulation environment but does not hinder the validation of the safety constraints.

B. Dynamic properties

The robot's equations of motion can be stated as in (1), namely:

$$\mathrm{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathrm{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) + \boldsymbol{\tau}_D(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\tau} + \boldsymbol{\tau}_e \quad (28)$$

Before the robot's joint accelerations in response to applied actuator torques can be determined it is necessary to be able to find the other components as functions of the joint position and velocity.

***Mass and inertia components**:* Each link is given the inertial properties of a cylinder with a length of $0.5\,\mathrm{m}$, radius of $5\,\mathrm{cm}$, and a mass of $5\,\mathrm{kg}$. The principal inertia frame of a link i is located $0.25\,\mathrm{m}$ in the z-direction above Ψ_i, or in other words, when $s = 0.5$ in (27).

The joint-space mass matrix $\mathrm{M}(\mathbf{q})$, torques due to centripetal and coriolis forces $\mathrm{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$ and torques due to gravity $\mathbf{g}(\mathbf{q})$ are all fully determined by the mass and rotational inertia of the links and the current configuration and velocity. The manipulator's motion is too complex for these components to be found analytically using MATLAB's Symbolic Toolbox, which is why its Robotics System Toolbox is used instead. This toolbox provides efficient functions for determining each of these components as well as the poses and geometric Jacobians of each frame. It requires a kinematic description of the robot with each link's mass and rotational inertia properties in URDF format.

***Joint friction**:* The model and coefficients of the friction of the robot's joints is based on the parameter identification of the KUKA LBR iiwa 14 R820 done in [29]. The KUKA iiwa 14 is a manipulator with size and total mass comparable to the manipulator used in this work and with the same sequence of joint axes. Thus the friction of its joints was deemed a reasonable estimate. In [29] the model used for the friction is:

$$\boldsymbol{\tau}_D(\dot{\mathbf{q}}) = \mathrm{D}_v\dot{\mathbf{q}} + \mathrm{D}_s\mathrm{sign}(\dot{\mathbf{q}}) \quad (29)$$

Where D_v and $\mathrm{D}_s \in \mathbb{R}^{n\times n}$ are diagonal matrices containing the viscous and coulomb friction parameters. The $\mathrm{sign}()$ operator returns a vector/matrix containing the sign of each element in the operand vector/matrix.

In [29] the physical friction parameters for each of the iiwa 14 joints were identified. While these parameters for the first six of its seven joints could be used directly, the links farther along the iiwa 14 are much smaller and lighter than those used in this simulation. Which is why the friction parameters of the iiwa 14's first joint are used for all joints acting in the z-axis. Similarly, the friction parameters of the second joint off the iiwa 14 are used for all joints acting in the x-axis. These first two links have a mass relatively close to those in this simulation. Hence, for this simulation the viscous friction parameters are:

$$\begin{aligned} \mathrm{D}_{vi} &= 0.241\,50\,\mathrm{N\,s\,m^{-1}} \text{ for } i = 1, 3, 5 \\ \mathrm{D}_{vi} &= 0.373\,28\,\mathrm{N\,s\,m^{-1}} \text{ for } i = 2, 4, 6 \end{aligned} \quad (30)$$

and the coulomb friction parameters are:

$$\begin{aligned} \mathrm{D}_{si} &= 0.319\,09\,\mathrm{N} \text{ for } i = 1, 3, 5 \\ \mathrm{D}_{si} &= 0.181\,30\,\mathrm{N} \text{ for } i = 2, 4, 6 \end{aligned} \tag{31}$$

Here i is used to indicate the position in the diagonal matrices.

C. Contact torques

The obstacle is modeled using the properties of the upper arm, and thus has a contact stiffness, k_h, of $30\,\mathrm{N\,mm^{-1}}$ [7]. For the contact radius, r_ϕ, $0.01\,\mathrm{m}$ was deemed a suitable estimate for the upper arm compression before bone is reached. No literature could be found stating an estimate for the damping due to the human upper arm during a collision. Since it is expected that some damping would be present a viscous damping parameter, D_ϕ, of $0.1\,\mathrm{N\,s\,m^{-1}}$ is chosen.

The force applied by the human obstacle to the robot is then formulated as:

$$\mathbf{F}_h = \left(k_h(r_\phi - d_{c\phi}) + D_\phi v_{c\phi}\right)\hat{\mathbf{u}}_{\phi c} \tag{32}$$

Where $d_{c\phi}$ is the distance between the contact point and the obstacle, $v_{c\phi}$ is the velocity of the contact point towards the obstacle, and $\hat{\mathbf{u}}_{\phi c}$ is the unit vector pointing from the obstacle to the contact point. The joint torques caused by this external force on robot are then determined using

$$\boldsymbol{\tau}_e = \mathrm{J}_{c,v}^\top \mathbf{F}_h \tag{33}$$

VI. Control

In this section it will be discussed how the robot described in section V is controlled to reach a target position while taking constraints into accounts. To do this a general overview of the entire simulation procedure is first provided. Then the method for determining the desired actuator torques for the robot to complete it task is discussed. Finally, the quadratic program which finds the optimal actuator torques to complete the task while adhering to constraints is discussed.

A. Overall simulation procedure

Before diving into the details of how the robot is controlled, it is useful to first look at an overview of the control and simulation process. A block diagram showing this overview is provided in figure 4.

Through a combination of an impedance controller and gravity compensation the desired torque vector for the robot's end-effector to reach a desired goal destination is computed. This is then is limited to not exceed the maximum torque the actuators can apply. A QP solver is then used to find the torque vector that minimizes its least-square error with the desired torque while complying to the safety constraints discussed in section IV. Since the QP solver finds an optimal torque for the actuators the safety constraints as stated in (24) and (25) are used in the solver. The torques computed by the quadratic program are then also limited to not exceed a specified physical actuator limit.

The actuator torques and the torques due to contact with the obstacle are then applied to the robot to determine the resulting joint accelerations using (21). The joint positions and velocities for the next discrete time are then determined using the discrete linear approximation given in (20) and (19). Once these are obtained the process can be repeated until the total time has been simulated.

B. Task control

***Operating principles**:* The primary controller used to complete the task is an impedance controller. This impedance controller can be thought of as a multidimensional spring with stiffness matrix $\mathrm{K} \in \mathbb{R}^{6\times6}$ between the end-effector frame with pose H_x^0 and another frame in space with pose H_f^0. This virtual spring applies a wrench at the end-effector in an attempt to make H_x^0 identical to H_f^0. The stiffness matrix of this spring is defined as [8]:

$$\mathrm{K} = \begin{bmatrix} \mathrm{K}_o & \mathrm{K}_c \\ \mathrm{K}_c^\top & \mathrm{K}_t \end{bmatrix} \tag{34}$$

Where K_o, K_t, and $\mathrm{K}_c \in \mathbb{R}^{3\times3}$ are the symmetric rotational, translational, and coupling stiffness matrices. For each of these stiffness matrices one can also define the corresponding co-stiffness matrices: G_o, G_t, and G_c. When the stiffness matrix is known the co-stiffness matrix can be found via [8]:

$$\mathrm{G}_i = \frac{1}{2}\mathrm{tr}(\mathrm{K}_i)\mathrm{I}_3 - \mathrm{K}_i \tag{35}$$

Where $\mathrm{tr}()$ is the tensor trace operator, and i is a placeholder for either o, t, or c. Beyond the stiffness matrix the wrench exerted by this spring is dependent on the relative pose between the two frames:

$$\mathrm{H}_x^f = \begin{bmatrix} \mathrm{R}_x^f & \mathbf{p}_x^f \\ \mathbf{0}_3^\top & 1 \end{bmatrix} \tag{36}$$

and its corresponding inverse:

$$\mathrm{H}_f^x = \begin{bmatrix} \mathrm{R}_f^x & \mathbf{p}_f^x \\ \mathbf{0}_3^\top & 1 \end{bmatrix} \tag{37}$$

Using these relative poses and the co-stiffness matrices the wrench exerted on the end-effector by the virtual spring expressed in its frame can be determined through [8]:

$$\tilde{\mathbf{m}}_S^x = -2\mathrm{as}(\mathrm{G}_o\mathrm{R}_x^f) - \mathrm{as}(\mathrm{G}_t\mathrm{R}_f^x\tilde{\mathbf{p}}_x^f\tilde{\mathbf{p}}_x^f\mathrm{R}_x^f) - 2\mathrm{as}(\mathrm{G}_c\tilde{\mathbf{p}}_x^f\mathrm{R}_x^f) \tag{38}$$

$$\tilde{\mathbf{F}}_S^x = -\mathrm{R}_f^x\mathrm{as}(\mathrm{G}_t\tilde{\mathbf{p}}_x^f)\mathrm{R}_x^f - \mathrm{as}(\mathrm{G}_t\mathrm{R}_f^x\tilde{\mathbf{p}}_x^f\mathrm{R}_x^f) - 2\mathrm{as}(\mathrm{G}_c\mathrm{R}_x^f) \tag{39}$$

Where $\mathrm{as}()$ is an operator that returns the skew-symmetric part of a square matrix. From $\tilde{\mathbf{m}}_S^x$ and $\tilde{\mathbf{F}}_S^x$ the components that make up the wrench on the end-effector, $\mathbf{W}_S^0$, can be directly found. This wrench is then transformed to be expressed in the inertial frame and translated into the joint-space of the robot:

$$(\mathbf{W}_S^0)^\top = \mathrm{Ad}_{\mathrm{H}_0^x}^\top(\mathbf{W}_S^x)^\top \tag{40}$$

$$\boldsymbol{\tau}_S = J_x^\top(\mathbf{W}_S^0)^\top \tag{41}$$

Therefore by applying an actuator torque of $\boldsymbol{\tau}_S$ the robot will move as if being pulled by this multidimensional spring.

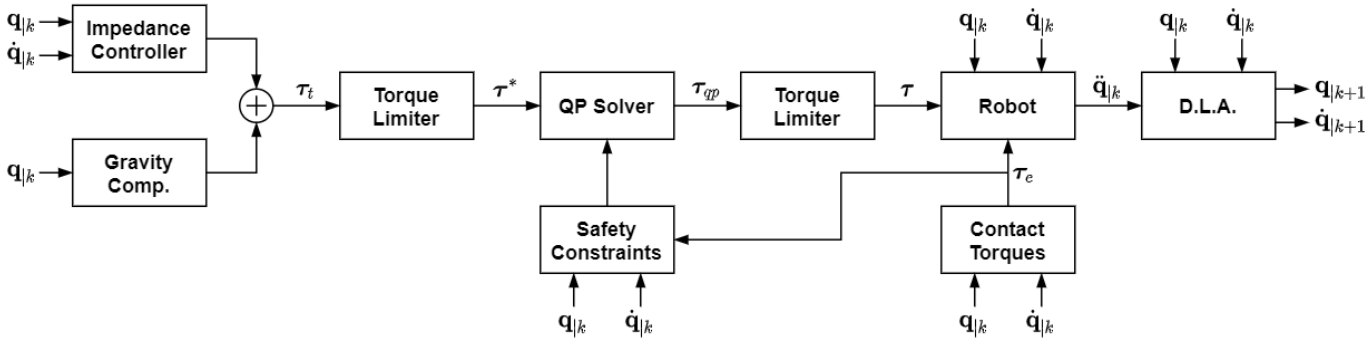


Fig. 4: Block diagram showing the control and simulation process for a time-step k.

In addition to this spring, an impedance controller can also add additional viscous damping to the robot to achieve the desired dynamic behavior. This damping can be implemented either via a multidimensional damper between the current and desired end-effector poses or by injecting additional damping on each of the joints directly. In this work it was chosen to inject the damping on the joints directly. Furthermore, a gravity compensation term is added so that this does not need to be done by the impedance controller's spring. The actuator torques for task completion are therefore computed as:

$$\boldsymbol{\tau}_t = \boldsymbol{\tau}_S - \mathrm{B}\dot{\boldsymbol{q}} + \boldsymbol{\tau}_g \tag{42}$$

Where $\mathrm{B} \in \mathbb{R}^{6\times6}$ is diagonal matrix containing the additional viscous damping coefficients applied to each joint and $\boldsymbol{\tau}_g$ is the torque vector required to compensate for gravity.

This torque could already be used as a desired torque for the quadratic program, however, then the physical limits of the actuators are not taken into account. An actuator always has a specified maximum torque that it can apply and so this will also be implemented on this manipulator. The desired torque given to the QP solver is then:

$$\boldsymbol{\tau}^* = \mathrm{sign}(\boldsymbol{\tau}_t)^\top \min(\boldsymbol{\tau}_{\max}, |\boldsymbol{\tau}_t|) \tag{43}$$

Where $\boldsymbol{\tau}_{\max}$ is a vector containing the maximum allowed torque for each joint and $\min()$ is an operator that performs an element-wise comparison between the operands and selects the lower value.

***Practical Implementation**:* During simulations the purpose of the impedance controller is to bring the end-effector from its initial position, $\mathbf{p}^0_{x,0}$, to a goal position $\mathbf{p}^0_{x,*}$. Only the postion of the end-effector is controlled to reach a goal, it is free reach the position in any orientation. Hence, the stiffness matrices of the impedance controller are set as

$$\mathrm{K}_t = 175\mathrm{I}_3,\ \mathrm{K}_o = 0\mathrm{I}_3,\ \mathrm{K}_c = 0\mathrm{I}_3 \tag{44}$$

The damping matrix of the impedance controller is set to

$$\mathrm{B} = 7.5\mathrm{I}_6 \tag{45}$$

The gravity compensation term is set to $\boldsymbol{\tau}_g = \mathbf{g}(\mathbf{q})$ such that the gravity terms cancel in (28).

When the determining the friction parameters in section V the values for the first two joints of the KUKA iiwa 14 were used. For this reason the torque limits of these joints will also be used. According to its datasheet both these joints have a torque limit of $320\,\mathrm{N\,m}$, and so each element of $\boldsymbol{\tau}_{\max}$ is set to 320.

Before the controller can be be used it is also necessary to define H^0_f. While it is possible to make this pose matrix constant and located at $\mathbf{p}^0_{x,*}$ this would result in very large torques being applied when the simulation begins. This is caused by the large extension of the virtual spring with almost no damping to mitigate it yet since joint velocities are initially zero. Instead the position of the other end of the spring, $\mathbf{p}^0_f$, is made to travel in a straight line from $\mathbf{p}^0_{x,0}$ to $\mathbf{p}^0_{x,*}$ over time. It is defined as:

$$\mathbf{p}^0_f(t) = \begin{cases} \mathbf{p}^0_{x,0} + \frac{t}{T^*}(\mathbf{p}^0_{x,*} - \mathbf{p}^0_{x,0}) & \text{for } t < T^* \\ \mathbf{p}^0_{x,*} & \text{for } t \geq T^* \end{cases} \tag{46}$$

Where t is the current simulation time and T^* is the total time in which $\mathbf{p}^0_f$ travels from $\mathbf{p}^0_{x,0}$ to $\mathbf{p}^0_{x,*}$. T^* is set to 2.5 seconds. To prevent relative rotation between H^0_x and H^0_f from playing a role, the rotation matrix of H^0_f is made the same as that of the end-effector.

C. Quadratic program

The goal of the quadratic program is to compute the torque vector that completes the manipulators task as best as possible while ensuring that the safety constraints are followed. Without the presence of obstacles the manipulator can complete its task by applying $\boldsymbol{\tau}^*$. Therefore we can state the objective function for the quadratic program as the least-square-error measure between $\boldsymbol{\tau}^*$ and the functions input:

$$\min_{\boldsymbol{\tau}_{qp}} ||\boldsymbol{\tau}^* - \boldsymbol{\tau}_{qp}||^2 + \epsilon||\boldsymbol{\tau}_{qp}||^2 \tag{47}$$

Where $\epsilon||\boldsymbol{\tau}_{qp}||^2$ with $\epsilon << 1$ has been added as a regularization task to minimize the norm of the computed torque vector.

In order to use this objective function it must be rewritten to match the format of (7). This is done as follows:

$$\begin{aligned}
&\min_{\boldsymbol{\tau}_{qp}} ||\boldsymbol{\tau}^* - \boldsymbol{\tau}_{qp}||^2 + \epsilon||\boldsymbol{\tau}_{qp}||^2 \\
&= \min_{\boldsymbol{\tau}_{qp}} (\boldsymbol{\tau}^{*\top} - \boldsymbol{\tau}_{qp}^\top)(\boldsymbol{\tau}^* - \boldsymbol{\tau}_{qp}) + \epsilon\boldsymbol{\tau}_{qp}^\top\boldsymbol{\tau}_{qp} \\
&= \min_{\boldsymbol{\tau}_{qp}} (1+\epsilon)\boldsymbol{\tau}_{qp}^\top\boldsymbol{\tau}_{qp} - 2\boldsymbol{\tau}^{*\top}\boldsymbol{\tau}_{qp} \\
&= \min_{\boldsymbol{\tau}_{qp}} \frac{1}{2}\boldsymbol{\tau}_{qp}^\top[2(1+\epsilon)\mathrm{I}_6]\boldsymbol{\tau}_{qp} + (-2\boldsymbol{\tau}^*)^\top\boldsymbol{\tau}_{qp} \qquad (48)
\end{aligned}$$

Therefore by setting $\mathrm{Q} = 2(1+\epsilon)\mathrm{I}_6$ and $\mathbf{f} = -2\boldsymbol{\tau}^*$ (as given in (7)) the objective function has been written in the QP format. In this derivation it is used that components not dependent on $\boldsymbol{\tau}_{qp}$ do not influence at which $\boldsymbol{\tau}_{qp}$ the function is minimal and that $\boldsymbol{\tau}^{*\top}\boldsymbol{\tau}_{qp} = \boldsymbol{\tau}_{qp}^\top\boldsymbol{\tau}^*$.

***Constraints**:* The only constraints that are imposed in the quadratic program are the safety constraints given in (24) and (25). For each obstacle there is one constraint for the impact energy and a contact force constraint for each link currently in contact. Each individual constraint i has the following structure:

$$\mathrm{A}_i\boldsymbol{\tau}_{qp} \leq b_i$$

Where $\mathrm{A}_i \in \mathbb{R}^{1\times 6}$ is a row vector and b_i a scalar. For the impact energy constraints

$$\begin{aligned}
\mathrm{A}_i &= \delta t\mathbf{J}_{\alpha\phi}\,\mathrm{M}^{-1} \\
b_i &= \sqrt{\frac{2E_{lim}}{m_{\alpha\phi}}} - \mathbf{J}_{\alpha\phi}(\dot{\mathbf{q}}_{|k} - \delta t\mathrm{M}^{-1}\,(\boldsymbol{\tau}_e - \boldsymbol{\gamma}))
\end{aligned}$$

and for contact force constraints

$$\begin{aligned}
\mathrm{A}_i &= \mathbf{J}_{c\phi,F} \\
b_i &= F_{safe}
\end{aligned}$$

The total number of constraints imposed in the quadratic program will vary in time depending on whether the approaching point at the current time for an obstacle and on how many links are currently in contact with an obstacle. However, regardless of the number of constraints the rows A_i and scalars b_i can always be stacked to form one large set of constraints. For instance, if there is one obstacle for which there is currently a approaching point on the robot and one link in contact then the quadratic program can be fully stated as:

$$\min_{\boldsymbol{\tau}_{qp}} \frac{1}{2}\boldsymbol{\tau}_{qp}^\top[2(1+\epsilon)\mathrm{I}_6]\boldsymbol{\tau}_{qp} + (-2\boldsymbol{\tau}^*)^\top\boldsymbol{\tau}_{qp} \qquad (49)$$

$$\text{s.t. } \mathrm{A}\boldsymbol{\tau}_{qp} \leq \mathbf{b} \qquad (50)$$

with

$$\mathrm{A} = \begin{bmatrix} \delta t\mathbf{J}_{\alpha\phi}\,\mathrm{M}^{-1} \\ \mathbf{J}_{c\phi,F} \end{bmatrix} \qquad (51)$$

$$\mathbf{b} = \begin{bmatrix} \sqrt{\frac{2E_{lim}}{m_{\alpha\phi}}} - \mathbf{J}_{\alpha\phi}(\dot{\mathbf{q}}_{|k} - \delta t\mathrm{M}^{-1}\,(\boldsymbol{\tau}_e - \boldsymbol{\gamma})) \\ F_{safe} \end{bmatrix} \qquad (52)$$

***Torque limit**:* After the QP solver has found the optimal joint torque vector $\boldsymbol{\tau}_{qp}$ it is first limited before being applied to the manipulator. This is done via:

$$\boldsymbol{\tau} = \mathrm{sign}(\boldsymbol{\tau}_{qp})^\top\min(\boldsymbol{\tau}_{\max}, |\boldsymbol{\tau}_{qp}|) \qquad (53)$$

The torque limit of the actuators could also have been imposed as a limit within the QP, however, then there is a risk that the problem could become infeasible due the safety constraints requiring torques larger than the limit. By limiting the torques in this way insight is gained into whether the physical limits in actuating the manipulator prevents the obstacle from being safe.

VII. Validation of safety constraints

In this section the results of the simulations described in section V are discussed. The constraints should always be able to keep E_{imp} below E_{lim} throughout the motion and $F_{c\phi}$ below F_{safe} in case of contact. The performance of the two constraints will be done separately as each has unique sources of error.

A. Test method

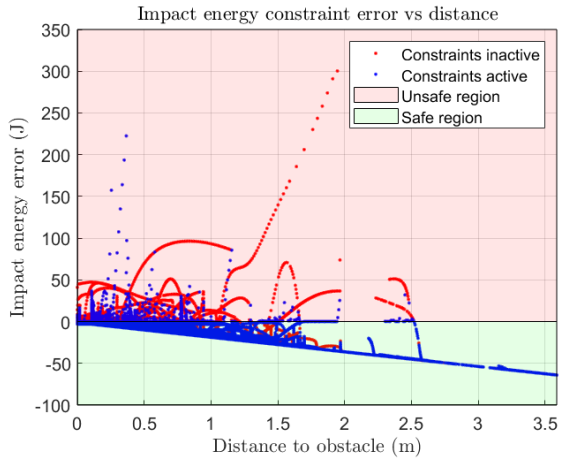
In order to test the effectiveness of the constraints in a variety of motions, 50 scenarios with randomized conditions are produced. For each scenario a random goal position is set and a random starting configuration that is at least $2\,\mathrm{m}$ away from its goal. Care was also taken that the goal position and the initial end-effector position are more than $1\,\mathrm{m}$ away from the base of the robot in the xy-plane. This is done to avoid beginning or ending in a singular configuration.

In each scenario the manipulator successfully reaches its goal within 5 seconds when there are no obstacles present. An obstacle is then placed such that it collides with a randomly chosen point along the manipulator at a randomly chosen time in its motion. The scenario with the obstacle in place is then simulated once without safety constraints and another time with safety constraints.

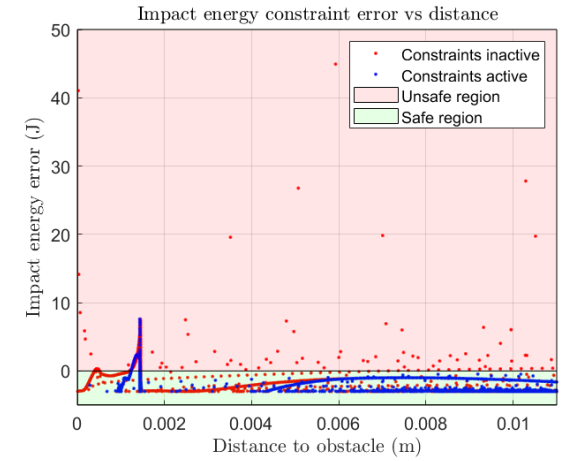
In each scenario the robot is simulated for a total of 5 seconds and the discrete time step size δt is 0.002 seconds. The safety constraints were configured as followed: $E_{safe} = 3\,\mathrm{J}$, $d_{safe} = 0.1\,\mathrm{m}$, $\kappa_E = 17.5\,\mathrm{J\,m^{-1}}$, and $F_{safe} = 150\,\mathrm{N}$.

B. Impact phase constraint

***Overall effect**:* Figure 5 shows the difference between the impact energy of the approaching point and the specified limit against the approaching point's distance to the obstacle for all 50 scenario's with and without the safety constraints active. The full range of data is shown in figure 5a while figure 5b is zoomed in to the cases when there is contact. In general it can be seen that when the constraints are used the kinetic energy of the approaching point less frequently becomes unsafe, however, significant errors are still made. This also confirmed by table I where it is shown that with the constraint active the energy limit is exceeded roughly half as often. The limit is still exceeded when the constraint is active, however, the average and standard deviation of the



(a) Impact energy constraint error against distance to obstacle.



(b) Impact energy constraint error against distance to obstacle, zoomed in to contact radius.

Fig. 5: Difference between impact energy of approaching point and current energy safety limit against distance to obstacle. The performance with and without the constraints active is shown and the regions where the obstacle is safe and unsafe are also indicated.

excess energy are both significantly lower when the constraints are active.

The optimal torques found by the QP solver are guaranteed to adhere to the constraints given. Therefore, any errors must arise from either changes to the parameters used in the constraint from one time-step to next or the physical limits of the actuators. In figure 5b a clear spike in the error is shown to be present regardless of whether the constraints are active. Such a spike could be caused by one significant source of error or a cluster of sources acting simultaneously. One would have to investigate the scenario in which the spike occurred to determine the cause for that specific case. Instead of analyzing the 50 scenarios individually, it was chosen to investigate all the scenarios at once in order to determine what are likely the most significant sources of error for an arbitrary run.

	Without constraints	With constraints
Positive error	6.2%	3.5%
Average positive error	8.39	4.03
Standard deviation of positive error	16.50	9.71

TABLE I: Table summarizing the effect of the impact energy constraint on the safety of an obstacle.

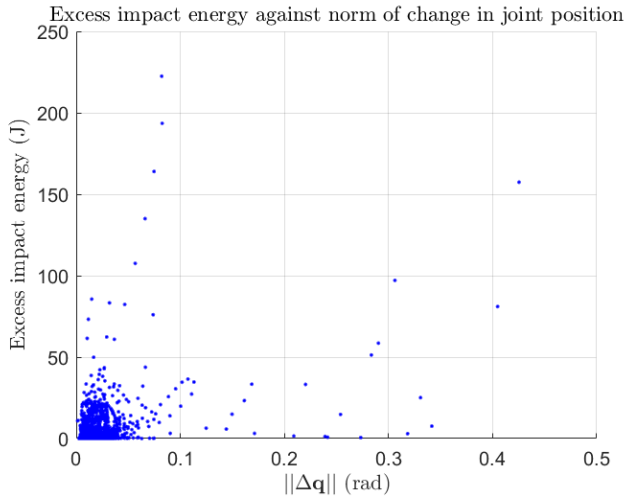


Fig. 6: Energy exceeding the safety limit against the norm of the change in joint position over the time-step.

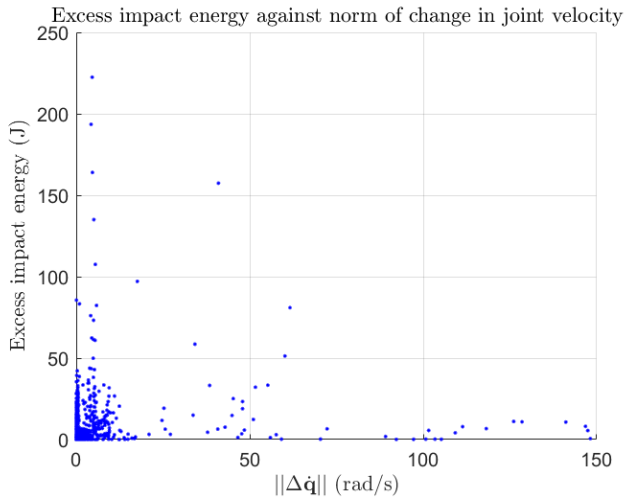


Fig. 7: Energy exceeding the safety limit against the norm of the change in joint velocity over the time-step.

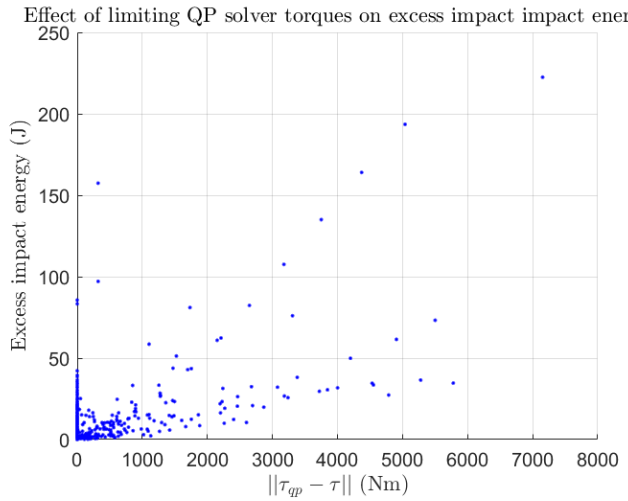


Fig. 8: Energy exceeding the safety limit against the norm of difference between the unlimited joint torques found by the QP solver and the the limited torques applied at the joints.

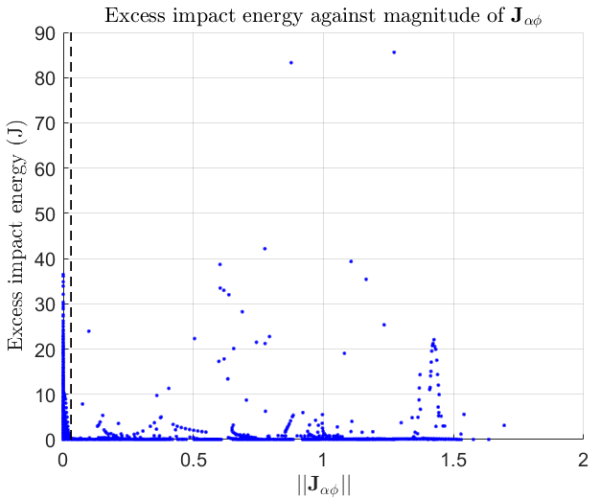


Fig. 9: Energy exceeding the safety limit against the magnitude of $\mathbf{J}_{\alpha\phi}$ when the torque found by the QP solver was not affected by the actuator limits. The black dashed line is $||\mathbf{J}_{\alpha\phi}|| = 0.03$

Joint position and velocity changes: In section IV it is discussed that, in order to formulate the constraints, the assumption must be made that the joint positions, velocities, and parameters dependent on them do not change significantly over time. Figures 6 and 7 show the unsafe errors made by the impact constraint (also called excess impact energy) against the magnitude of the change in joint position and velocity, respectively.

In figure 6 one can see that there are a handful of cases where we see large changes in joint position having a relatively large excess energy but one can also see instances of low error at similarly large joint position differences. The largest constraint errors are made when the joint position changes comparatively little. These points, with excess energy above $100\,\mathrm{J}$ and smaller than $0.1\,\mathrm{rad}$ joint position difference magnitude, do still exhibit a slight trend of increasing with joint position difference. The lack of a clear relationship between large changes in joint position and the error made by the constraints suggests that it is not a dominant cause for the large errors made by the constraint. However, the change in joint position does introduce some error and it is possible that there are conditions under which the constraint is more sensitive to changes in joint position.

Figure 7 does not show any significant relationship between the change in joint velocity over a time-step and the error made by the constraint. This can be seen by the fact that despite there being a broad range for the change in velocity there is no trend showing an increase in excess energy with large changes in velocity.

Actuator torque limit: One possible source of error is the torque limiter placed between the torques computed by the QP solver and those applied to the robot joints. Figure 8 shows the relation between how significantly the torques found by the QP solver are limited and the error made by the constraint.

In this figure it can be seen that, in general, as the difference between $\boldsymbol{\tau}_{qp}$ and $\boldsymbol{\tau}$ increases the error by made the constraint also increases. This indicates that one reason why the robot cannot adhere to the impact energy constraint is that the physical limits of the actuators do not allow for the necessary torque to be applied to dissipate the approaching point's excess energy within a single time-step. One can also see distinctly different trends in how the excess energy increases with increased difference between optimal and applied torque. These trends are determined by the current configuration and velocity of the robot as well as its control task, causing the energy of the approaching point to be dissipated more quickly in some cases than others. It was decided not to investigate these specific cases further in order to focus on identifying other causes of error for the constraint.

In figure 8 one can also see a vertical line of points at zero, where the torque found by the solver is not limited at all. This confirms that the physical limit of the actuators is not the only source of error present. Although this was already known, investigating the cases where the optimal torque was not physically limited can give further insight into the generally most significant sources of error.

Effective mass of the robot: One apparent source of error present in the simulation when investigating the cases where the QP solver was not limited by actuator limits was found to be the magnitude of $\mathbf{J}_{\alpha\phi}$, defined in (12). Figure 9 shows the excess energy against the magnitude of $\mathbf{J}_{\alpha\phi}$ while the optimal torque was within actuator limits. In this figure one can see that the constraint often makes errors when the magnitude of this vector is nearly zero. This is caused by the effective mass of the approaching point becoming extremely large in these cases. The relationship between the effective mass and the magnitude of $\mathbf{J}_{\alpha\phi}$ is shown in figure 10.

By looking at (10) the reason for this relationship between the effective mass and $\mathbf{J}_{\alpha\phi}$ becomes quite clear. The effec-

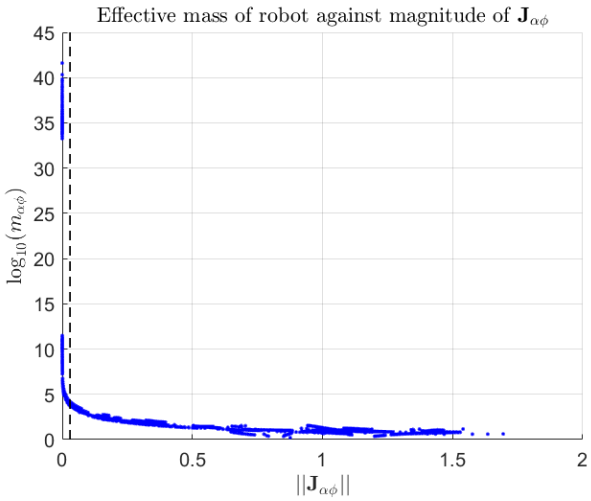


Fig. 10: Effective mass of approaching point toward obstacle against the magnitude of $\mathbf{J}_{\alpha\phi}$ when the torque found by the QP solver was not affected by the actuator limits. The black dashed line is $||\mathbf{J}_{\alpha\phi}|| = 0.03$

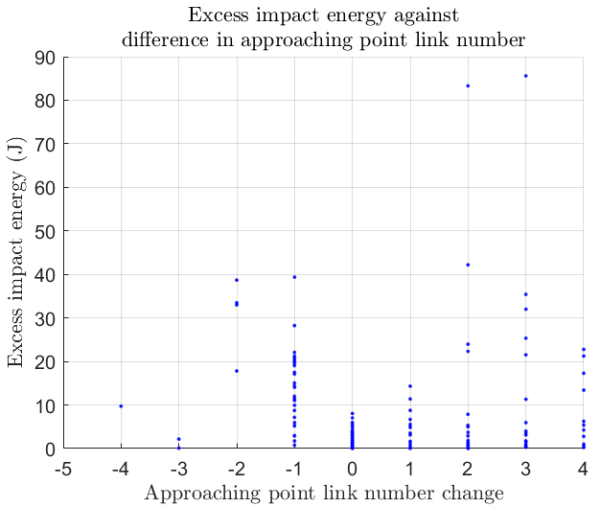


Fig. 11: Excess energy of the approaching point against the change in link the approaching point is on. A value of 1 means that the approaching was on link i the previous time-step but is now on link $i+1$. Only cases where the optimal torque is not limited and $||\mathbf{J}_{\alpha\phi}|| > 0.03$ are shown.

tive mass has a roughly inverse square relationship with the components of $\mathbf{J}_{\alpha\phi}$ and so when the values in $\mathbf{J}_{\alpha\phi}$ are very small, $m_{\alpha\phi}$ becomes very large. Beyond the effective mass reaching extremely large values, figure 10 also shows that the effective mass becomes very sensitive to changes in the magnitude of $\mathbf{J}_{\alpha\phi}$ which is to be expected considering their inverse square relation. The effective mass is then also very sensitive to changes in $\mathbf{q}$ since it determines $\mathbf{J}_{\alpha\phi}$. This leads to the impact energy constraint being very sensitive to small changes in joint position which can result in large errors being made.

As shown in (12), $\mathbf{J}_{\alpha\phi}$ is composed of the unit vector pointing from the approaching point to the obstacle and the Jacobian relating the joint velocities to the Cartesian velocity of the approaching point. Its magnitude can be small due to the Cartesian velocity of the approaching point being nearly perpendicular to the unit vector toward the obstacle. The magnitude of $\mathbf{J}_{\alpha\phi}$ can also be small due to the Jacobian having zero and near zero entries. This could be the case if the approaching point is found to be near the base of the second link for instance. These points do have a velocity but a very small one and only affected by the first two joints.

Change in approaching point link number: Each time-step the approaching point on the robot is found again. Between two time-steps the approaching point could in principle switch from being at the end-effector to being halfway along the second link. This can have a large impact on the constraint since the distance to the obstacle, its effective mass, and velocity can all change significantly due to this. The effect of such a switch is entirely dependent on the specific case. For example, if the approaching point is moved to a point farther away and moving slowly it appears to have a positive effect. It is also possible that while limiting the energy of the approaching point other parts of the robot must move quickly to complete the task. If such a part then comes closer to the obstacle and becomes the approaching point it suddenly has a significant amount of energy that must be dissipated.

Figure 11 shows the change in the link number the approaching point is on against the excess energy of the approaching point. In this figure only cases where the optimal torque is not limited and $||\mathbf{J}_{\alpha\phi}|| > 0.03$ are shown in order to filter out the points related to these already identified sources of error. In this figure it can be seen that the largest remaining errors occur when the approaching point has switched to a different link from where it was the previous time-step. At the same time, it is also noted that the small errors made by the constraint also occur in these cases. As was described before a change in link number for the approaching point is not necessarily detrimental. However, it increases the risk of error as it increases the chance that the parameters used in the constraint in one time-step are not even remotely similar to those used in the next.

C. Quasi-static contact constraint

For the analysis of the quasi-static contact constraint scenario 50 was removed from the data set. In this scenario the obstacle was placed on top of the second joint causing the first and second link to always be in contact with the obstacle. With the contact points located at or very near the second joint of the manipulator the geometric Jacobian for the contact points is entirely populated with zero and near-zero values (order of 10^{-5} and lower). The pseudoinverse of such a geometric Jacobian then contains very large values (order of 10^4 and greater) which leads to the calculation of extremely large contact forces ranging from being in the order of 10^5 to the order of 10^{17}. Therefore, this scenario is considered an

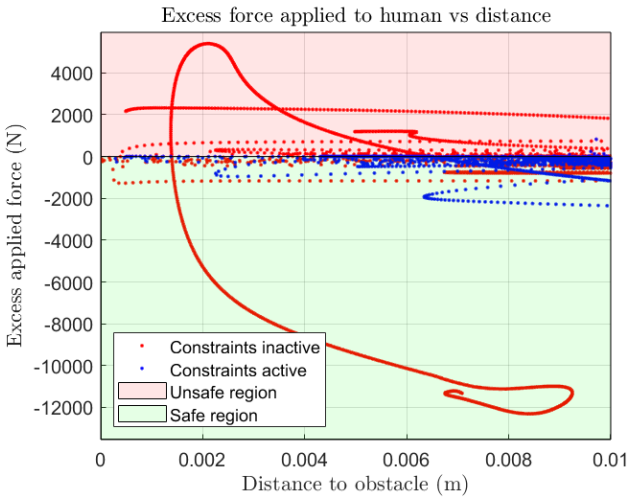


Fig. 12: Difference between applied force and F_{safe} against the distance to the obstacle. The performance with and without the constraints active is shown and the regions where the obstacle is safe and unsafe are also indicated.

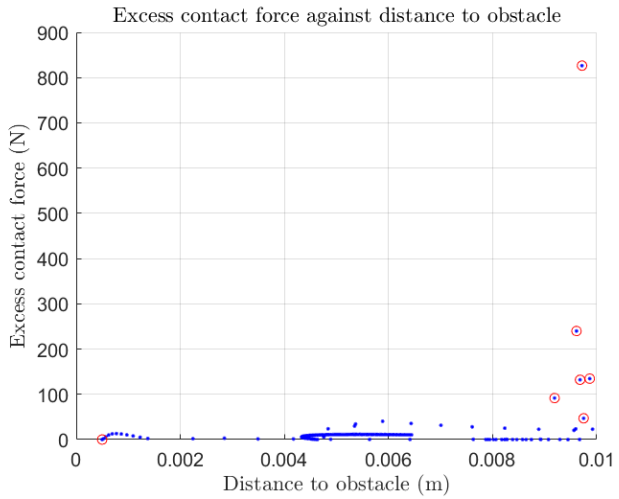


Fig. 13: Excess force applied by the robot against the distance to the obstacle. A red circle around a point indicates that that point is a first contact point.

	Without constraints	With constraints
Number of scenarios with contact	49	27
Total seconds of contact across all scenarios	9.722	3.326
Positive error	25.6%	14.3%
Average positive error	1.9×10^3	14.2
Standard deviation of positive error	1.8×10^3	56.8

TABLE II: Table summarizing the effect of the contact force constraint on the safety of an obstacle.

anomaly for the purposes of analyzing the performance of the contact force constraint.

Overall effect: Figure 12 shows difference between the force applied by the robot and F_{safe} against the distance to the obstacle for the remaining 49 scenario's with and without the constraint active. The figure clearly shows that implementation of the constraints reduces the forces applied by the robot effectively in general. However, even with the constraints active the applied force does still exceed F_{safe} at times.

The overall effect of the constraint is summarized in table II. In this table it can be seen that when the constraints are active roughly half the scenarios no longer collide with the obstacle. This is due to the manipulator finding other paths to complete its task due to the constraint on the impact energy of the approaching point. The average collision time in a scenario with contact is also reduced from $0.2\,\mathrm{s}$ without constraints to $0.12\,\mathrm{s}$ with constraints. Table II also shows that the contact force exceeds the limit roughly half as often with the constraints active. Furtermore, the constraints drastically reduces the average force applied exceeding the limit as well as the standard deviation. All in all, it can be said that the constraints have a positive effect on the safety of the obstacle in terms of the force applied to it. However, the constraint cannot be said to guarantee the safety of the obstacle so the remainder of this section will identify the causes of the contact force exceeding its limit despite the constraint.

First contact: Figure 13 shows the contact forces that exceeded the safety limit with the constraints active with the first contact points indicated with red circles around them. One of these first contact points is very close to the obstacle. In the scenario this occurred the obstacle was placed such that it was in contact with the manipulator at the start of the simulation time. With the exception of this point, the figure shows that the most significant errors correspond to the first time contact is detected. This is before the contact force constraint is put into effect since that is only done after contact is detected. One might expect that this could be caused by the physical limits of the actuators or the impact energy exceeding its limit. However, the optimal torque is not limited nor does the impact energy exceed its limit for any of these points, first contact or not.

The reason these occur is thought to be because, the potential energy in the controller is not fully controlled. The impact energy constraint controls the kinetic energy of the approaching point and the potential energy of the controller indirectly as it is converted into kinetic energy. When contact is first detected the actuator torques are optimized to limit the kinetic energy of the contact point including the converted potential energy, but the remaining potential energy of the controller is not constrained yet and is then free to exert a large force for one time-step before this is limited by the contact force constraint.

Changes in joint position: Just as with the impact energy constraint, in order to formulate the contact force constraint the assumption had to be made that the joint positions would not change significantly from one time-step to the next. In order

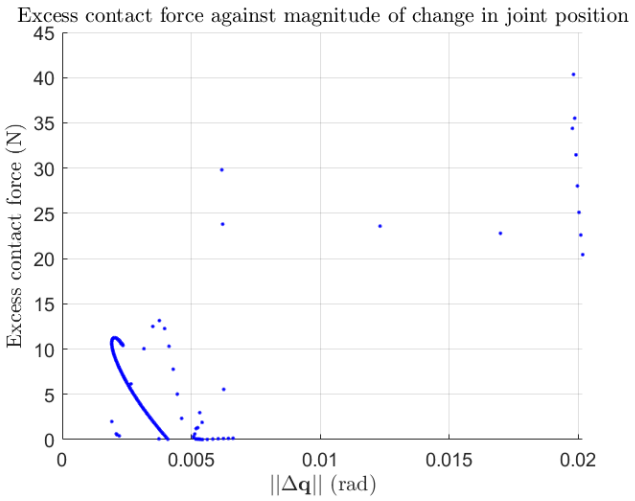


Fig. 14: Excess force applied by the robot against the magnitude of the change in joint position.

to evaluate the impact of this assumption the contact forces in excess of the safety limit are shown against the change in joint position in figure 14. The change in joint position induces error due to difference in the parameters used in the constraint between the two time-steps. The first contact points are therefore not shown here since constraint was not active yet.

Figure 14 shows that as the change in joint position over a time-step increases the error made by the constraint also increases as one would expect. There are a few cases where low error and high error occur for similar changes in joint position. Both motion of the manipulator and the computation of the parameters used in the constraint are highly non-linear so a certain amount of inconsistencies are to be expected. Furthermore it also possible that the same change in joint position caused a similar increase in contact force but the previous contact force was further below the limit. This increase would then raise the force to just above the limit instead of roughly $20\,\mathrm{N}$ above it.

VIII. Conclusion

In conclusion, this work has attempted to introduce human safety concepts to control methods used by multi-tasking robots such as humanoids. This was done by first investigating the behavior of a human-robot collision and what the relevant measures are for determining the severity of an injury. Then a safety criteria was proposed for each phase of a collision between robot and human. The first criteria is based on limiting the kinetic energy of the nearest approaching point on the robot toward the obstacle. The second criteria is based on limiting the applied crushing force by the robot to the obstacle after contact has been detected. The criteria were then transformed into QP-compatible linear inequality constraints for the robot that constrain either the joint torques or accelerations.

The performance of the joint torque version of the constraints was tested in simulation. It was shown that the constraints significantly improve the safety of an obstacle placed in the manipulator's path compared to having no safety constraints. The constraints cannot guarantee safety, however, as energies and forces exceeding their limits were still present. For the energy-based constraint the errors are attributed to a combination of joint position changes over a single time-step, physical limits of the actuator, the effective robot mass nearing a singularity, and a change in the link the approaching point is on. For the contact force-based constraint the errors are attributed to a combination of being the first instance of contact and changes in the joint position over a single time-step.

Improvements to these constraints should focus on improving their robustness. One way this can be done is by considering the impact energy of each link rather than the whole robot to prevent the approaching point position from changing too much. By only considering points with a large enough $\mathbf{J}_{\alpha\phi}$ magnitude singularities of the effective mass could be avoided. The force applied at the first instance of contact can be limited by activating this constraint a small distance before contact is established rather than once contact is established. Finally, a smaller time-step size would reduce errors due to changes in joint positions over time.

Beyond these improvements other future work can be done to develop better safety constraints for multi-tasking robots. Future work that would contribute to the development of these constraints is a deeper investigation into how the distance-based energy limit function could be determined based on, for instance, the mass and rotational inertia of the manipulator links. It is also worth investigating whether a single carefully chosen safety criteria can ever sufficiently guarantee harm from all forms of injury.

References

[1] F. Aller, D. Pinto-Fernandez, D. Torricelli, J. L. Pons, and K. Mombaur, "From the state of the art of assessment metrics toward novel concepts for humanoid robot locomotion benchmarking," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 914–920, 2020.

[2] C. Vogel, M. Fritzsche, and N. Elkmann, "Safe human-robot cooperation with high-payload robots in industrial applications," in *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2016, pp. 529–530.

[3] D. Araiza-Illan and A. de San Bernabé Clemente, "Dynamic regions to enhance safety in human-robot interactions," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018, pp. 693–698.

[4] A. M. Zanchettin, B. Lacevic, and P. Rocco, "A novel passivity-based control law for safe human-robot coexistence," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2012, pp. 2276–2281.

[5] C. Byner, B. Matthias, and H. Ding, "Dynamic speed and separation monitoring for collaborative robot applications – concepts and performance," *Robotics and Computer-Integrated Manufacturing*, vol. 58, pp. 239 – 252, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S073658451830259X

[6] S. Haddadin, S. Haddadin, A. Khoury, T. Rokahr, S. Parusel, R. Burgkart, A. Bicchi, and A. Albu-Schäffer, "A truly safely moving robot has to know what injury it may cause," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2012, pp. 5406–5413.

[7] N. Lucci, B. Lacevic, A. M. Zanchettin, and P. Rocco, "Combining speed and separation monitoring with power and force limiting for safe collaborative robotics applications," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6121–6128, 2020.

[8] G. Raiola, C. A. Cardenas, T. S. Tadele, T. de Vries, and S. Stramigioli, "Development of a safety- and energy-aware impedance controller for collaborative robots," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1237–1244, April 2018.

[9] N. Mansfeld, B. Djellab, J. R. Veuthey, F. Beck, C. Ott, and S. Haddadin, "Improving the performance of biomechanically safe velocity control for redundant robots through reflected mass minimization," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 5390–5397.

[10] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka, "The development of honda humanoid robot," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, vol. 2, 1998, pp. 1321–1326 vol.2.

[11] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi, and T. Isozumi, "Humanoid robot hrp-2," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 2, 2004, pp. 1083–1090 Vol.2.

[12] B. Henze, M. A. Roa, and C. Ott, "Passivity-based whole-body balancing for torque-controlled humanoid robots in multi-contact scenarios," *The International Journal of Robotics Research*, vol. 35, no. 12, pp. 1522–1543, 2016. [Online]. Available: https://doi.org/10.1177/0278364916653815

[13] S. Dafarra, G. Romualdi, G. Metta, and D. Pucci, "Whole-body walking generation using contact parametrization: A non-linear trajectory optimization approach," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 1511–1517.

[14] Y. Lee, N. Tsagarakis, and J. Lee, "Agile standing-up control of humanoids: Energy-based reactive contact wrench optimization with strict dynamic consistency," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 4652–4659.

[15] B. Henze, A. Dietrich, and C. Ott, "An approach to combine balancing with hierarchical whole-body control for legged humanoid robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 700–707, 2016.

[16] T. Koolen, S. Bertrand, G. Thomas, T. De Boer, T. Wu, J. Smith, J. Englsberger, and J. Pratt, "Design of a momentum-based control framework and application to the humanoid robot atlas," *International Journal of Humanoid Robotics*, vol. 13, pp. 1 650 007–1, 03 2016.

[17] M. Zafar, S. Hutchinson, and E. A. Theodorou, "Hierarchical optimization for whole-body control of wheeled inverted pendulum humanoids," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 7535–7542.

[18] A. Meguenani, V. Padois, and P. Bidaud, "Control of robots sharing their workspace with humans: An energetic approach to safety," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 4678–4684.

[19] A. Meguenani, V. Padois, J. Da Silva, A. Hoarau, and P. Bidaud, "Energy based control for safe human-robot physical interaction," in *2016 International Symposium on Experimental Robotics*, D. Kulić, Y. Nakamura, O. Khatib, and G. Venture, Eds. Springer International Publishing, 2017, pp. 809–818.

[20] U. S. Park, Y. Yamada, and Y. Nakabo, "Force control with safety constraints via iterative feedback tuning," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 3670–3675.

[21] B. Ponton, A. Herzog, S. Schaal, and L. Righetti, "A convex model of humanoid momentum dynamics for multi-contact motion generation," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 2016, pp. 842–849.

[22] S. Haddadin and E. Croft, *Physical Human–Robot Interaction*. Springer International Publishing, 2016, pp. 1835–1874. [Online]. Available: https://doi.org/10.1007/978-3-319-32552-1_69

[23] T. S. Tadele, T. de Vries, and S. Stramigioli, "The safety of domestic robotics: A survey of various safety-related publications," *IEEE Robotics Automation Magazine*, vol. 21, no. 3, pp. 134–142, 2014.

[24] L. Thollon, Y. Godio, S. Bidal, and C. Brunet, "Evaluation of a new security system to reduce thoracic injuries in case of motorcycle accidents," *International Journal of Crashworthiness*, vol. 15, no. 2, pp. 191–199, 2010. [Online]. Available: https://doi.org/10.1080/13588260903102062

[25] M. Laffranchi, N. G. Tsagarakis, and D. G. Caldwell, "Safe human robot interaction via energy regulation control," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 35–41.

[26] N. Gupta, J. Smith, B. Shrewsbury, and B. Børnich, "2d push recovery and balancing of the ever3 - a humanoid robot with wheel-base, using model predictive control and gain scheduling," in *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, 2019, pp. 365–372.

[27] J. Vorndamme, M. Schappler, and S. Haddadin, "Collision detection, isolation and identification for humanoids," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4754–4761.

[28] D. Popov and A. Klimchik, "Real-time external contact force estimation and localization for collaborative robot," in *2019 IEEE International Conference on Mechatronics (ICM)*, vol. 1, 2019, pp. 646–651.

[29] Y. R. Stürz, L. M. Affolter, and R. S. Smith, "Parameter identification of the kuka lbr iiwa robot including constraints on physical feasibility," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 6863 – 6868, 2017, 20th IFAC World Congress. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405896317317147

APPENDIX A
A BRIEF INTRODUCTION TO USED SCREW THEORY

Throughout this work screw theory is used to model to the robot, the interaction between the robot and the obstacle, set the safety criteria, and determine the torque due to the impedance controller. This appendix is intended to introduce the key concepts of screw theory used in the main body of this work.

A. Poses

Consider two frames in space denoted with Ψ_i and Ψ_0. The pose of a frame describes its position and orientation in space with respect to another frame. The pose of Ψ_i with respect to the inertial frame Ψ_0 is defined as:

$$\mathrm{H}_i^0 = \begin{bmatrix} \mathrm{R}_i^0 & \mathbf{p}_i^0 \\ \mathbf{0}_3^\top & 1 \end{bmatrix} \tag{54}$$

Where $\mathrm{R}_i^0 \in SO(3)$ is a rotation matrix describing the orientation of Ψ_i with respect to Ψ_0, $\mathbf{p}_i^0 \in \mathbb{R}^3$ is a vector describing the location of Ψ_i with respect to Ψ_0, and $\mathbf{0}_3^\top$ is a 3-element row vector of zeros. This additional row of three zero's and a one allows for pose matrices to be easily used to change the reference frame in which a point a in space is expressed. If the position of a point $\mathbf{p}$ is known with respect to the frame Ψ_i this is denoted with $\mathbf{p}^i$. The position of $\mathbf{p}$ with respect to the frame Ψ_0, $\mathbf{p}^0$, can then be found as:

$$\mathbf{P}^0 = \mathrm{H}_i^0 \, \mathbf{P}^i \tag{55}$$

Where $\mathbf{P}^0$ and $\mathbf{P}^i \in \mathbb{R}^4$ are known as the homogeneous coordinates of $\mathbf{p}^0$ and $\mathbf{p}^i$ respectively. They are defined as:

$$\mathbf{P}^0 \equiv \begin{bmatrix} \mathbf{p}^0 \\ 1 \end{bmatrix}, \, \mathbf{P}^i \equiv \begin{bmatrix} \mathbf{p}^i \\ 1 \end{bmatrix} \tag{56}$$

Dual to the pose matrix H_i^0 is the pose matrix H_0^i which expresses the location and orientation of Ψ_0 with respect to Ψ_i:

$$\mathrm{H}_0^i = (\mathrm{H}_i^0)^{-1} = \begin{bmatrix} (\mathrm{R}_i^0)^\top & -(\mathrm{R}_i^0)^\top \mathbf{p}_i^0 \\ \mathbf{0}_3^\top & 1 \end{bmatrix} = \begin{bmatrix} \mathrm{R}_0^i & \mathbf{p}_0^i \\ \mathbf{0}_3^\top & 1 \end{bmatrix} \tag{57}$$

Note that the product of a pose matrix and its inverse yields the identity matrix. Now consider that there is a third frame in the space, Ψ_j, and that the pose of Ψ_j with respect to Ψ_i, H_j^i, is known. The pose of Ψ_j with respect to Ψ_0 can be obtained through:

$$\mathrm{H}_j^0 = \mathrm{H}_i^0 \, \mathrm{H}_j^i \tag{58}$$

B. Twists

A twist is the generalization of velocity for a rigid body and describes both the rotational and translational motion of a body. Consider a frame Ψ_i attached to a moving rigid body such that a point a on the body is fixed with respect to Ψ_i. Now consider another frame Ψ_j that is not attached to the body. The velocity of the point a expressed in Ψ_j can then be written as:

$$\begin{aligned}
\dot{\mathbf{P}}^j &= \dot{\mathrm{H}}_i^j \, \mathbf{P}^i \\
\dot{\mathbf{P}}^j &= \dot{\mathrm{H}}_i^j \, \mathrm{H}_j^i \, \mathbf{P}^j \\
\dot{\mathbf{P}}^j &= \begin{bmatrix} \dot{\mathrm{R}}_i^j & \dot{\mathbf{p}}_i^j \\ \mathbf{0}_3^\top & 0 \end{bmatrix} \begin{bmatrix} \mathrm{R}_j^i & \mathbf{p}_j^i \\ \mathbf{0}_3^\top & 1 \end{bmatrix} \mathbf{P}^j \\
\dot{\mathbf{P}}^j &= \begin{bmatrix} \dot{\mathrm{R}}_i^j \, \mathrm{R}_j^i & \dot{\mathrm{R}}_i^j \, \mathbf{p}_j^i + \dot{\mathbf{p}}_i^j \\ \mathbf{0}_3^\top & 0 \end{bmatrix} \mathbf{P}^j \\
\dot{\mathbf{P}}^j &= \begin{bmatrix} \dot{\mathrm{R}}_i^j \, \mathrm{R}_j^i & \dot{\mathbf{p}}_i^j - \dot{\mathrm{R}}_i^j \, \mathrm{R}_j^i \, \mathbf{p}_i^j \\ \mathbf{0}_3^\top & 0 \end{bmatrix} \mathbf{P}^j
\end{aligned} \tag{59}$$

Since R_i^j belongs to the group $SO(3)$ it is known that $\dot{\mathrm{R}}_i^j \mathrm{R}_j^i$ is a skew-symmetric matrix that can be written in the form:

$$\dot{\mathrm{R}}_i^j \, \mathrm{R}_j^i = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \tag{60}$$

The skew-matrix contains three unique values meaning it can also be constructed from the elements of a vector. The following notation is therefore introduced:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \in \mathbb{R}^3 \Rightarrow \tilde{\mathbf{x}} \equiv \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \tag{61}$$

In the case of the screw-symmetric matrix $\dot{\mathrm{R}}_i^j \mathrm{R}_j^i$ the equivalent vector that can construct this matrix represents the generalized angular velocity of Ψ_i with respect to Ψ_j expressed in Ψ_j. It is written as:

$$\boldsymbol{\omega}_i^{j,j} \in \mathbb{R}^3 \Rightarrow \tilde{\boldsymbol{\omega}}_i^{j,j} = \dot{\mathrm{R}}_i^j \, \mathrm{R}_j^i \tag{62}$$

Using this, the velocity of the point $\mathbf{p}$ can be written as:

$$\begin{aligned}
\dot{\mathbf{P}}^j &= \begin{bmatrix} \tilde{\boldsymbol{\omega}}_i^{j,j} & \dot{\mathbf{p}}_i^j - \tilde{\boldsymbol{\omega}}_i^{j,j} \, \mathbf{p}_i^j \\ \mathbf{0}_3^\top & 0 \end{bmatrix} \mathbf{P}^j \\
\dot{\mathbf{P}}^j &= \begin{bmatrix} \tilde{\boldsymbol{\omega}}_i^{j,j} & \dot{\mathbf{p}}_i^j + \mathbf{p}_i^j \times \boldsymbol{\omega}_i^{j,j} \\ \mathbf{0}_3^\top & 0 \end{bmatrix} \mathbf{P}^j \\
\dot{\mathbf{P}}^j &= \begin{bmatrix} \tilde{\boldsymbol{\omega}}_i^{j,j} & \mathbf{v}_i^{j,j} \\ \mathbf{0}_3^\top & 0 \end{bmatrix} \mathbf{P}^j
\end{aligned} \tag{63}$$

Where $\mathbf{v}_i^{j,j}$ represents the translational velocity of Ψ_i with respect to Ψ_j expressed in Ψ_j. This matrix contains 6 unique values and so can also be represented through:

$$\mathbf{T}_i^{j,j} = \begin{bmatrix} \boldsymbol{\omega}_i^{j,j} \\ \mathbf{v}_i^{j,j} \end{bmatrix} \in \mathbb{R}^6 \Rightarrow \tilde{\mathbf{T}}_i^{j,j} \equiv \begin{bmatrix} \tilde{\boldsymbol{\omega}}_i^{j,j} & \mathbf{v}_i^{j,j} \\ \mathbf{0}_3^\top & 0 \end{bmatrix} \tag{64}$$

$\mathbf{T}_i^{j,j}$ is called the twist of Ψ_i with respect to Ψ_j expressed in Ψ_j. Another twist can also be found by rewriting the expression for $\dot{\mathbf{P}}^j$ in a different manner, namely:

$$\begin{aligned}
\dot{\mathbf{P}}^j &= \dot{\mathrm{H}}_i^j \, \mathbf{P}^i \\
\dot{\mathbf{P}}^j &= \mathrm{H}_i^j \, \mathrm{H}_j^i \, \dot{\mathrm{H}}_i^j \, \mathbf{P}^i \\
\dot{\mathbf{P}}^j &= \mathrm{H}_i^j \begin{bmatrix} \mathrm{R}_j^i & \mathbf{p}_j^i \\ \mathbf{0}_3^\top & 1 \end{bmatrix} \begin{bmatrix} \dot{\mathrm{R}}_i^j & \dot{\mathbf{p}}_i^j \\ \mathbf{0}_3^\top & 0 \end{bmatrix} \mathbf{P}^i \\
\dot{\mathbf{P}}^j &= \mathrm{H}_i^j \begin{bmatrix} \mathrm{R}_j^i \, \dot{\mathrm{R}}_i^j & \mathrm{R}_j^i \, \dot{\mathbf{p}}_i^j \\ \mathbf{0}_3^\top & 0 \end{bmatrix} \mathbf{P}^i \\
\dot{\mathbf{P}}^j &= \mathrm{H}_i^j \begin{bmatrix} \mathrm{R}_j^i \, \dot{\mathrm{R}}_i^j & \mathrm{R}_j^i \, (-\dot{\mathrm{R}}_i^j \, \mathbf{p}_j^i) \\ \mathbf{0}_3^\top & 0 \end{bmatrix} \mathbf{P}^i \\
\dot{\mathbf{P}}^j &= \mathrm{H}_i^j \begin{bmatrix} \mathrm{R}_j^i \, \dot{\mathrm{R}}_i^j & -\mathrm{R}_j^i \, \dot{\mathrm{R}}_i^j \, \mathbf{p}_j^i - \dot{\mathbf{p}}_j^i \\ \mathbf{0}_3^\top & 0 \end{bmatrix} \mathbf{P}^i \\
\dot{\mathbf{P}}^j &= \mathrm{H}_i^j \begin{bmatrix} \tilde{\boldsymbol{\omega}}_i^{i,j} & -\tilde{\boldsymbol{\omega}}_i^{i,j} \, \mathbf{p}_j^i - \dot{\mathbf{p}}_j^i \\ \mathbf{0}_3^\top & 0 \end{bmatrix} \mathbf{P}^i \\
\dot{\mathbf{P}}^j &= \mathrm{H}_i^j \begin{bmatrix} \tilde{\boldsymbol{\omega}}_i^{i,j} & \tilde{\mathbf{v}}_i^{i,j} \\ \mathbf{0}_3^\top & 0 \end{bmatrix} \mathbf{P}^i \\
\dot{\mathbf{P}}^j &= \mathrm{H}_i^j \, \tilde{\mathbf{T}}_i^{i,j} \, \mathbf{P}^i
\end{aligned} \tag{65}$$

$\mathbf{T}_i^{i,j}$ is called the twist of Ψ_i with respect to Ψ_j expressed in Ψ_i. Through these derivations it can be seen that a twist describes both the rotations and translations between two frames over time. If these frames are each attached to a rigid body it turns out that the twist between the two frames will be the same regardless of where on the rigid bodies the frames are attached. Due to this twists can also describe the rotation and translation of an entire rigid body relative to another rigid body expressed in a reference frame.

C. Wrenches

It is known that the power of a purely translating body is:

$$\text{Power} = \mathbf{F} \, \mathbf{v} \tag{66}$$

Where $\mathbf{F} \in \mathbb{R}^3$ is the applied force as a row vector (co-vector), and $\mathbf{v} \in \mathbb{R}^3$ is its velocity as a column vector. Since the twist is the generalization of velocity there must an analogous vector for the generalization of its forces. This generalization of force is known as a wrench and it is defined as:

$$\mathbf{W}^i = \begin{bmatrix} \mathbf{m}^i & \mathbf{F}^i \end{bmatrix} \tag{67}$$

Where $\mathbf{m}^i$ represents the torques applied to a body, $\mathbf{F}^i$ represents the linear forces applied to the body, and i indicates the frame the wrench is expressed in. The total power of a rigid body due to translation and rotation can then be computed if the

body's twist and wrench are expressed in the same frame. In the case of a body a, its power expressed in the inertial frame Ψ_0 is found as:

$$\text{Power} = \mathbf{W}_a^0 \, \mathbf{T}_a^{0,0} \tag{68}$$

D. Adjoints

Adjoints are matrices that facilitate the change of reference frame a twist or wrench is expressed in. As one can imagine, changing the reference frame will be dependent on the pose of the of the current reference frame relative to the new reference frame. Consider the twist of body a relative to a body b expressed in the frame Ψ_i: $\mathbf{T}_a^{i,b}$. In order to express the twist between these two bodies in another reference frame Ψ_j we write:

$$\mathbf{T}_a^{j,b} = \text{Ad}_{\text{H}_i^j} \, \mathbf{T}_a^{i,b} \tag{69}$$

Where $\text{Ad}_{\text{H}_i^j} \in \mathbb{R}^{6\times 6}$ is the Adjoint of H_i^j and it defined as:

$$\text{Ad}_{\text{H}_i^j} \equiv \begin{bmatrix} \text{R}_i^j & 0 \\ \tilde{\mathbf{p}}_i^j \, \text{R}_i^j & \text{R}_i^j \end{bmatrix} \tag{70}$$

Changing the reference frame in which a wrench is expressed is done through:

$$(\mathbf{W}^i)^\top = \text{Ad}_{\text{H}_i^j}^\top \, (\mathbf{W}^j)^\top \tag{71}$$

APPENDIX B
USED CODE

In this appendix the MATLAB code used to generate the results is provided. First the two primary files are presented. In the first the 50 scenario conditions are found, and the second simulates each of each scenarios while applying the safety constraints. Afterwards the supporting functions created are presented and finally the urdf file with the manipulator description is shown. Any used functions not listed in this appendix are functions native to MATLAB 2020a or its Robotics Systems Toolbox.

A. Primary scripts

1) FindScenarios.m: The MATLAB script used to generate the conditions of the 50 scenarios is as follows:

```
clear all
close all
clc
addpath('Kinematics');
addpath('ScrewTheory');
addpath('SafetyFunctions');
addpath('urdf');

tStart = tic;
% Load robot model
robot = importrobot('myrobot.urdf');
robot.DataFormat = 'column';
robot.Gravity = [0;0;-9.81];
N = robot.NumBodies-1;

% Set homogeneous joint damping
% Friction is dependent on joint axis
BvisX = 0.37328; BvisZ = 0.24150; % Viscous friction coefficients
Bvis = diag([BvisZ; BvisX; BvisZ; BvisX; BvisZ; BvisX]);
BcoulX = 0.18130; BcoulZ = 0.31909; % Coulomb friction coefficients
Bcoul = diag([BcoulZ; BcoulX; BcoulZ; BcoulX; BcoulZ; BcoulX]);
% Define friction torques as function of joint velocity
B = @(dq) Bvis*dq + Bcoul*sign(dq);

% Set total simulation time and stepsize
T = 5;
dt = 0.002;
ITER = T/dt;

% Set Control parameters
Kp = 175*eye(3);
Kd = 7.5*eye(N);
TtoGoal = 2.5; % Time in which end of spring goes from pEE0 to goalPos
maxTau = 320; % Maximum allowed torque, based on KUKA iiwa 14
e = 0.01;

ro = 0.01; % Radius of collision sphere.
%% Find scenarios
% In each iteration of the while loop, we generate a random end-effector
% goal and a randomized starting configuration. We then simulate the motion
% and determine if the manipulator successfully reached its goal by looking
% at the distance between the end-effector and the goal of the last time
% discrete times. If successful a obstacle is placed at a random point
% along a random link at a random time. The first link is not included
% since it only rotates in place.
```

```
numContactPaths = 0;
maxContactPaths = 50; % Total number of paths to find
% We store the found initial configuration and goal position pair and the
% position of the obstacle in space. We also store some other data related
% to the obstacle placement in case insight into this is desired during
% further analysis.
q0Contact = zeros(6,maxContactPaths); % Initial config
qContact = zeros(6,maxContactPaths); % Config at contact
nContact = zeros(1,maxContactPaths); % Link of contact
sContact = zeros(1,maxContactPaths); % Position on link of contact
goalPosContact = zeros(3,maxContactPaths); % Goal Position
oContact = zeros(3,maxContactPaths); % Obstacle position
tContact = zeros(1,maxContactPaths); % Time of contact

while numContactPaths < maxContactPaths
    disp(['Starting new scenario. Number of paths found so far: ', ...
        num2str(numContactPaths)]);
    % Make arrays to store q, dq, ddq, end-effector position and control
    % error
    q = zeros(N, ITER+1);
    dq = zeros(N, ITER+1);
    ddq = zeros(N, ITER+1);

    pEE = zeros(3, ITER+1);
    % These are for determining if the target is successfully reached.
    d_error = zeros(1,ITER+1); % EE distance to target
    vEE_goal = zeros(1,ITER+1); % EE velocity to target

    % Set motion target
    % Bounds for goal position so that it is reachable
    gp_maxCoXY = 2.5;
    gp_minCoXY = -2.5;
    gp_maxCoZ = 2.5;
    gp_minCoZ = 0;
    gp_rmin = 1.5;
    gp_rmax = 2.5;

    goalPos = zeros(3,1);
    gp_ok = false;
    while ~gp_ok
        goalPos(1:2) = (gp_maxCoXY - gp_minCoXY).*rand(2,1) + gp_minCoXY;
        goalPos(3) = (gp_maxCoZ - gp_minCoZ).*rand(1,1) + gp_minCoZ;

        % Avoid goals close to base in xy-plane to limit chance for
        % singular configurations.
        if (norm(goalPos(1:2)) > 1.0)
            gp_r = norm(goalPos);
            if (gp_r > gp_rmin && gp_r < gp_rmax)
                gp_ok = true;
            end
        end
    end

    % Determine initial configuration
    d0 = 0;
    d0_min = 2;
```

```
q0_max = pi;
q0_min = -pi;
while d0 < d0_min
    q0 = (q0_max-q0_min).*rand(N,1) + q0_min;
    HEE0 = getTransform(robot, q0, robot.BodyNames{end});
    pEE0 = HEE0(1:3,4);
    if norm(pEE0(1:2)) > 1.0 % Avoid initial EE positions close to base
        d0 = norm(goalPos - pEE0);
    end
end
q(:,1) = q0;

%% Simulate motion
for i = 1:1:ITER+1
    M = massMatrix(robot,q(:,i));
    invM = M\eye(N);
    G = gravityTorque(robot,q(:,i));
    Cdq = velocityProduct(robot,q(:,i),dq(:,i));
    beta = Cdq + G + B(dq(:,i));

    % Get kinematics of robot at current time
    Hi = getJointPoses(robot, q(:,i));
    pn = getJointPositions(Hi);
    Jn = getJointJacobians(robot,q(:,i));

    pEE(:,i) = pn(:,end); % End-effector position
    d_error(i) = norm(goalPos-pEE(:,i)); % Distance to goal
    uEE_goal = (goalPos-pEE(:,i))/d_error(i);
    vEE_goal(i) = uEE_goal.'*Jn{7}(4:6,:)*dq(:,i);

    % Impedance control to goalPos
    t = (i-1)*dt;
    % Set current position of spring end-point
    if t < TtoGoal
        pf = pEE0 + (t/TtoGoal)*(goalPos - pEE0);
    else
        pf = goalPos;
    end
    Hf = Hi{7}; % To give spring end the same R as the EE
    Hf(1:3,4) = pf;

    HEE_f = inverseH(Hf)*Hi{7}; % EE pose relative to goal
    REE_f = HEE_f(1:3,1:3);
    pEE_f = HEE_f(1:3,4);

    mtilde = -as(0.5*Kp*REE_f.'*tilde(pEE_f)*tilde(pEE_f)*REE_f);
    ftilde = -REE_f.'*as(0.5*Kp*tilde(pEE_f))*REE_f ...
        - as(0.5*Kp*REE_f.'*tilde(pEE_f)*REE_f);
    WEE = [mtilde(3,2); mtilde(1,3); mtilde(2,1); ...
        ftilde(3,2); ftilde(1,3); ftilde(2,1)];
    W0 = Adjoint(inverseH(Hi{7})).'*WEE;

    tauVS = Jn{7}.'*W0; % Torque due to virtual spring
    tauVD = -Kd*dq(:,i); % Additional joint damping
    tauImp = tauVS + tauVD;
    tauDes = tauImp + G; % Desired actuator torque
```

```
        % Cap desired torques to known limits
        tauDes = sign(tauDes).*min(maxTau,abs(tauDes));

        % Do QP optimization
        Q = 2*(1+e)*eye(N);
        f = -2*tauDes;

        A = [];
        b = [];

        options = optimset('Display', 'off'); % Suppress solver messages
        % Determine optimal torque input using QP solver
        tauOpt = quadprog(Q,f,A,b,[],[],[],[],[],options);
        % Cap tauOpt according to actuator torque limit
        tauOpt = sign(tauOpt).*min(maxTau,abs(tauOpt));

        % Determine joint acceleration as sum of homogeneous dynamics
        % and control torques
        ddq(:,i) = invM*(tauOpt - beta);
        % Determine joint position and velocity after step by assuming
        % ddq is constant throughout step.
        dq_next = dq(:,i) + dt*ddq(:,i);
        q_next = q(:,i) + dt*dq(:,i) + (dt^2/2)*ddq(:,i);

        if i ~= ITER+1
            q(:,i+1) = q_next;
            dq(:,i+1) = dq_next;
        end
    end

    % Last 10 instances are within 0.01m of the target with low velocity it
    % is considered a successfull run.
    if (sum(d_error(end-9:end)<=0.01) == 10 && ...
            sum(vEE_goal(end-9:end)<=0.01) == 10)
        % Choose random time interval
        i_c = randi(ITER+1);
        % Choose random link that isnt 1 since that can only rotate in
        % place
        n_c = randi([2,6]);
        % Choose random point along link to place obstacle
        s_c = rand;
        % Get joint positions at time i_c
        pn_c = getJointPositions(getJointPoses(robot, q(:,i_c)));
        o = pn_c(:,n_c) + s_c*(pn_c(:,n_c+1) - pn_c(:,n_c));

        % Just be sure, check if collision occurs
        contact = checkForCollisions(pn_c, o, ro);
        if contact
            numContactPaths = numContactPaths + 1;
            q0Contact(:,numContactPaths) = q0; % Initial config
            qContact(:,numContactPaths) = q(:,i_c); % Initial config
            nContact(numContactPaths) = n_c; % link of contact
            sContact(numContactPaths) = s_c; % Position on link of contact
            goalPosContact(:,numContactPaths) = goalPos; % Goal Position
            oContact(:,numContactPaths) = o; % Obstacle position
            tContact(numContactPaths) = (i_c-1)*dt; % Time of contact
```

```
                disp( Valid path found!!! );
            end
        end
end
% Save data for use in RunScenarionsWithSafety.m
save( ContactScenarios.mat , numContactPaths , q0Contact , qContact , ...
      nContact , sContact , goalPosContact , oContact , tContact , ...
      Kp , Kd , B );

% Display total simulation time in minutes in the command window after
% completion.
totaltime = toc(tStart);
total_minutes = totaltime/60
```

2) RunScenariosWithSafety.m The MATLAB script used to run all found scenarios while considering safety of the obstacle is:

```
clear all
close all
clc
addpath( ScrewTheory );
addpath( SafetyFunctions );
addpath( Kinematics );
addpath( urdf );
% Load scenario conditions
load ContactScenarios.mat;

tStart = tic;
% Load robot model
robot = importrobot( myrobot.urdf );
robot.DataFormat = column ;
robot.Gravity = [0;0;-9.81];
N = robot.NumBodies-1;

% Set homogeneous joint damping
% Friction is dependent on joint axis
BvisX = 0.37328; BvisZ = 0.24150; % Viscous friction coefficients
Bvis = diag([BvisZ; BvisX; BvisZ; BvisX; BvisZ; BvisX]);
BcoulX = 0.18130; BcoulZ = 0.31909; % Coulomb friction coefficients
Bcoul = diag([BcoulZ; BcoulX; BcoulZ; BcoulX; BcoulZ; BcoulX]);
% Define friction torques as function of joint velocity
B = @(dq) Bvis * dq + Bcoul * sign(dq);

% Set total simulation time and stepsize
T = 5;
dt = 0.002;
ITER = T/dt;

% Set Control parameters
Kp = 175 * eye(3);
Kd = 7.5 * eye(N);
TtoGoal = 2.5; % Time in which end of spring goes from pEE0 to goalPos
maxTau = 320; % Maximum allowed torque, based on KUKA iiwa 14
e = 0.01;

% Distance based energy limit
Edmin = 0.1;
```

```
Edmax = 0.5;
Esafe = 3;
Emax = 10;
kElim = (Emax - Esafe)/(Edmax - Edmin);
fElim = @(d) (d <= Edmin)*Esafe + (d > Edmin)*(Esafe + kElim*(d-Edmin));

% Contact force limit
Fsafe = 150;

% Create cells to store data from each run for later analysis. Since
% simulating all 50 scenarios takes a long time, a lot of data is stored in
% case insight into them is desired later.

% Load mat to store data in.
load ResultsWithSafety;

nFailed = 0; % Track number of failed scenarios
for run = 41:1:50
    disp(['Starting run ', num2str(run)]);
    % Configuration variables
    q = zeros(N, ITER+1);
    dq = zeros(N, ITER+1);
    ddq = zeros(N, ITER+1);
    tauAct = zeros(N,ITER+1); % Applied torque
    tauOpt = zeros(N,ITER+1); % Torque found by solver
    invM = cell(1,ITER+1);
    pEE = zeros(3, ITER+1);

    q(:,1) = q0Contact(:,run);
    % Save initial end-effector position
    HEE0 = getTransform(robot, q(:,1), robot.BodyNames{end});
    pEE0 = HEE0(1:3,4);

    % Task variables
    goalPos = goalPosContact(:,run);
    derror = zeros(1,ITER+1);

    % Eimp safety tracking variables
    Eimp = -1*ones(1,ITER+1);
    Elim = -1*ones(1,ITER+1);
    pa = zeros(3,ITER+1);
    dao = -1*ones(1,ITER+1);
    vao = -1*ones(1,ITER+1);
    Jao = cell(1,ITER+1);
    na = zeros(1,ITER+1);
    sa = -1*ones(1,ITER+1);

    % Contact dynamics variables
    o = oContact(:,run);
    ro = 0.01;
    kc = 30e3;
    CI = cell(N,1); % Contact Info
    tauc = zeros(N,ITER+1);

    % Applied force tracking variables
    Fc = zeros(N,ITER+1);
```

```
    Fh = zeros(N,ITER+1);
    pc = cell(N,ITER+1);
    doc = zeros(N,ITER+1);
    vco = zeros(N,ITER+1);
    uoc = cell(N,ITER+1);
    Jc = cell(N,ITER+1);
    invJcT = cell(N,ITER+1);

%     try
      for i = 1:1:ITER+1
          M = massMatrix(robot,q(:,i));
          invM{i} = M\eye(N);
          G = gravityTorque(robot,q(:,i));
          Cdq = velocityProduct(robot,q(:,i),dq(:,i));
          beta = Cdq + G + B(dq(:,i));

          % Get kinematics of robot at current time
          Hi = getJointPoses(robot, q(:,i));
          pn = getJointPositions(Hi);
          Jn = getJointJacobians(robot,q(:,i));

          pEE(:,i) = pn(:,end); % End-effector position
          derror(i) = norm(goalPos - pEE(:,i));

          % Search for collision points, determine force applied by robot
          % on human, determine force applied by human on robot
          CI = findCollisionInfo(pn, Jn, o, ro);
          if ~isempty(CI)
              nCI = length(CI);
              for c = 1:1:nCI
                  % Store contact data
                  pc{CI(c).nl,i} = CI(c).pc;
                  uoc{CI(c).nl,i} = CI(c).uoc;
                  doc(CI(c).nl,i) = CI(c).doc;
                  % We do -CI(c).uoc since uoc points from the obstacle
                  % to the robot.
                  vco(CI(c).nl,i) = -CI(c).uoc.'*CI(c).Jc*dq(:,i);
                  Jc{CI(c).nl,i} = CI(c).Jc;
                  invJcT{CI(c).nl,i} = CI(c).invJcT;
                  % Force applied at this time is due to torque computed
                  % and applied over previous time-step.
                  if i ~= 1
                      Fc(CI(c).nl,i) = -CI(c).uoc.'*CI(c).invJcT*tauAct(:,i-1);
                  else
                      % At t = 0 no applied torque yet
                      Fc(CI(c).nl,i) = 0;
                  end

                  F_HS = kc*(ro - CI(c).doc)*CI(c).uoc;

                  if (vco(CI(c).nl,i) > 0)
                      F_HD = 0.1*vco(CI(c).nl,i)*CI(c).uoc;
                  else
                      F_HD = [0;0;0];
                  end
                  Fh(CI(c).nl,i) = norm(F_HS + F_HD);
```

```
            % Add contact torque to total
            tauc(:,i) = tauc(:,i) + CI(c).Jc.'*(F_HS+F_HD);
        end
    end
    % Determine impact energy to obstacle
    DI = findDangerInfo(pn, Jn, dq(:,i), invM{i}, o);
    if ~isempty(DI)
        Eimp(i) = DI(1).E;
        Elim(i) = fElim(DI(1).d);
        pa(:,i) = DI(1).p;
        dao(i) = DI(1).d;
        vao(i) = DI(1).vu;
        Jao{i} = DI(1).J;
        na(i) = DI(1).nl;
        sa(i) = DI(1).s;
    end

    % Impedance control to goalPos
    t = (i-1)*dt;
    if t < TtoGoal
        pf = pEE0 + (t/TtoGoal)*(goalPos - pEE0);
    else
        pf = goalPos;
    end
    Hf = Hi{7}; % To give spring end the same R as the EE
    Hf(1:3,4) = pf;

    HEE_f = inverseH(Hf)*Hi{7}; % EE pose relative to goal
    REE_f = HEE_f(1:3,1:3);
    pEE_f = HEE_f(1:3,4);

    mtilde = -as(0.5*Kp*REE_f.'*tilde(pEE_f)*tilde(pEE_f)*REE_f);
    ftilde = -REE_f.'*as(0.5*Kp*tilde(pEE_f))*REE_f ...
        - as(0.5*Kp*REE_f.'*tilde(pEE_f)*REE_f);
    WEE = [mtilde(3,2); mtilde(1,3); mtilde(2,1); ...
        ftilde(3,2); ftilde(1,3); ftilde(2,1)];
    W0 = Adjoint(inverseH(Hi{7})).'*WEE;

    tauVS = Jn{7}.'*W0; % Torque due to virtual spring
    tauVD = -Kd*dq(:,i); % Additional joint damping
    tauImp = tauVS + tauVD; % Torques from impedance control
    tauDes = tauImp + G; % Desired actuator torque
    % Cap desired torques to known limits
    tauDes = sign(tauDes).*min(maxTau,abs(tauDes));

    % Do QP optimization
    Q = 2*(1+e)*eye(N);
    f = -2*tauDes;

    A = [];
    b = [];

    % Set safety constraints: Impact energy and contact force
    % limits
    if ~isempty(DI)
        AE = dt*DI(1).J*invM{i};
```

```
                bE = sqrt(2*Elim(i)/DI(1).m) - DI(1).J*(dq(:,i) ...
                    + dt*invM{i}*(tauc(:,i) - beta));

                A = [A; AE];
                b = [b; bE];
            end

            if ~isempty(CI)
                nCI = length(CI);
                for c = 1:1:nCI
                    AF = -CI(c).uoc.'*CI(c).invJcT;
                    bF = Fsafe;

                    A = [A; AF];
                    b = [b; bF];
                end
            end

            options = optimset('Display', 'off'); % Suppress solver messages

            % Determine optimal torque input using QP solver
            tauOpt(:,i) = quadprog(Q,f,A,b,[],[],[],[],[],options);
            % Cap tauOpt according to actuator torque limit
            tauAct(:,i) = sign(tauOpt(:,i)).*min(maxTau,abs(tauOpt(:,i)));

            % Determine joint acceleration as sum of homogeneous dynamics, control
            % torques, and contact torques.
            ddq(:,i) = invM{i}*(tauAct(:,i) + tauc(:,i) - beta);
            % Determine joint position and velocity after step by assuming
            % ddq is constant throughout step.
            dq_next = dq(:,i) + dt*ddq(:,i);
            q_next = q(:,i) + dt*dq(:,i) + (dt^2/2)*ddq(:,i);

            if i ~= ITER+1
                q(:,i+1) = q_next;
                dq(:,i+1) = dq_next;
            end

%             prog = i/(ITER)*100;
%             if floor(prog) == prog
%                 disp(['Simulation progress: ', num2str(i/ITER*100), '%']);
%             end
        end

        % Store all data from run
        qAllRuns{run} = q;
        dqAllRuns{run} = dq;
        ddqAllRuns{run} = ddq;
        tauActAllRuns{run} = tauAct;
        tauOptAllRuns{run} = tauOpt;
        taucAllRuns{run} = tauc;
        invMAllRuns{run} = invM;
        derrorAllRuns{run} = derror;

        ElimAllRuns{run} = Elim;
        EimpAllRuns{run} = Eimp;
```

```
        paAllRuns{run} = pa;
        daoAllRuns{run} = dao;
        vaoAllRuns{run} = vao;
        JaoAllRuns{run} = Jao;
        naAllRuns{run} = na;
        saAllRuns{run} = sa;

        FcAllRuns{run} = Fc;
        FhAllRuns{run} = Fh;
        pcAllRuns{run} = pc;
        docAllRuns{run} = doc;
        vcoAllRuns{run} = vco;
        uocAllRuns{run} = uoc;
        JcAllRuns{run} = Jc;
        invJcTAllRuns{run} = invJcT;
%     catch e
%         disp(['An error occured with identifier: ',e.identifier]);
%         disp(['The error message is: ',e.message]);
%         nFailed = nFailed+1;
%     end
end

save('ResultsWithSafety.mat', 'qAllRuns', 'dqAllRuns', 'ddqAllRuns', ...
    'tauActAllRuns', 'tauOptAllRuns', 'taucAllRuns', 'invMAllRuns', ...
    'derrorAllRuns', 'ElimAllRuns', 'EimpAllRuns', 'paAllRuns', ...
    'daoAllRuns', 'vaoAllRuns', 'JaoAllRuns', 'naAllRuns', 'saAllRuns', ...
    'FcAllRuns', 'FhAllRuns', 'pcAllRuns', 'docAllRuns', 'vcoAllRuns', ...
    'uocAllRuns', 'JcAllRuns', 'invJcTAllRuns');

simtime = toc(tStart);
sim_hours = simtime/3600;
disp(['Completed in ', num2str(sim_hours), ' hours. ', num2str(nFailed), ...
    ' scenarios failed.']);
```

B. Safety functions

The functions within the SafetyFunctions path are:

1) findDangerInfo.m:

```
function allDangerInfo = findDangerInfo(pn, Jn, dq, invM, o)
% This function determines which point on the robot, if any, is closest to
% an obstacle with a positive velocity toward it. It then outputs
% information related to this point.
%%% Inputs:
% pn: Positions of all frames on the robot, which are the link starting
% points and the end-effector. Each column is a set of (x,y,z) coordinates.
% Jn: Geometric Jacobians of each of these frames. dq: Joint velocity
% vector.
% invM: Inverse joint-space mass matrix.
% o: Position of the obstacles in the world frame. Each column is a set of
% (x,y,z) coordinates.
%%% Output:
% AllDangerInfo: array of Danger Information structs, one for each
% obstacle, containing information about its corresponding point alpha on
% the robot.

G = size(o,2); % Number of obstacles
```

```
N = length(dq); % Number of links

TJoints = zeros(6,N+1); % Twists of the frames at pn
vJoints = zeros(3,N+1); % Velocities of the frames at pn
for i = 1:1:N+1
    TJoints(:,i) = Jn{i}*dq;
    vJoints(:,i) = TJoints(4:6,i);
end

vu_N = zeros(G,N+1); % Components of v_N in direction of obstacle
smin_N = zeros(1,N); % Position along each link closest to obstacle that is moving
   toward it
dmin_N = zeros(1,N); % Smallest distance between obstacle and link part moving toward
    it.

allDangerInfo = [];

for m = 1:1:G
    % Initialize
    vu_N(m,1) = velToObst(pn(:,1),vJoints(:,1),o(:,m));
    for n = 1:1:N
        % Find velocity of endpoint on other side
        vu_N(m,n+1) = velToObst(pn(:,n+1),vJoints(:,n+1),o(:,m));

        if (vu_N(m,n) > 0 && vu_N(m,n+1) > 0)
            % Entire link poses danger since both ends have a positive
            % velocity to the obstacle. We then find the point along the
            % link that is closest to the obstacle.
            [smin_n, dmin_n] = minDistToObst(pn(:,n), pn(:,n+1), o(:,m), [0 1]);
            smin_N(n) = smin_n;
            dmin_N(n) = dmin_n;
        elseif (vu_N(m,n) <= 0 && vu_N(m,n+1) > 0)
            % Part of link poses danger. The start of the link has negative
            % velocity to the obstacle while the end of the link has a
            % positive velocity. So we first find where on the link this
            % velocity switches to positive. Then we find the closest point
            % to the obstacle of this part of the link.
            sSwitch = vuSwitchPoint(pn(:,n), vJoints(:,n), pn(:,n+1), vJoints(:,n+1),
                o(:,m));
            [smin_n, dmin_n] = minDistToObst(pn(:,n), pn(:,n+1), o(:,m), [sSwitch 1])
                ;
            smin_N(n) = smin_n;
            dmin_N(n) = dmin_n;
        elseif (vu_N(m,n) > 0 && vu_N(m,n+1) <= 0)
            % Part of link poses danger. The start of the link has positive
            % velocity to the obstacle while the end of the link has a
            % negative velocity. So we first find where on the link this
            % velocity switches to positive. Then we find the closest point
            % to the obstacle of this part of the link.
            sSwitch = vuSwitchPoint(pn(:,n), vJoints(:,n), pn(:,n+1), vJoints(:,n+1),
                o(:,m));
            [smin_n, dmin_n] = minDistToObst(pn(:,n), pn(:,n+1), o(:,m), [0 sSwitch])
                ;
            smin_N(n) = smin_n;
            dmin_N(n) = dmin_n;
        elseif (vu_N(m,n) <= 0 && vu_N(m,n+1) <= 0)
```

```
                % Link poses no danger. Both ends have a negative velocity
                % toward the obstacle so its impossible for any point between
                % these to have a positive velocity. We mark this links
                % distance and position with Inf.
                smin_N(n) = Inf;
                dmin_N(n) = Inf;
            end
        end
        % Find the link with the closest distance to the obstacle.
        [dmin, idx] = min(dmin_N);
        if (dmin == Inf)
            % Entire robot is moving away from obstacle so skip.
            continue;
        end
        % Find position and velocity of the alpha for this obstacle.
        smin = smin_N(idx);
        ps = pn(:,idx) + smin      *(pn(:,idx+1) - pn(:,idx));
        vs = vJoints(:,idx) + smin         *(vJoints(:,idx+1) - vJoints(:,idx));
        % Find unit vector pointing from alpha to obstacle.
        u = (o(:,m) - ps)/dmin;
        % Find velocity of alpha toward obstacle.
        vu = u.   *vs;

        % Store information related to alpha for use in other code.
        DangerInfo.obst = o(:,m); % Corresponding obstacle position
        DangerInfo.no = m; % Obstacle number
        DangerInfo.d = dmin; % Distance to obstacle
        DangerInfo.p = ps; % position of alpha in world frame
        DangerInfo.nl = idx; % Link number that alpha is on
        DangerInfo.s = smin_N(idx); % Alpha s position along link
        DangerInfo.u = u; % unit vector toward obstacle from alpha.
        DangerInfo.vu = vu; % velocity of alpha toward obstacle

        % Determine jacobian of alpha as weighted combination of jacobians for
        % the ends of the link.
        Js = Jn{idx}(4:6,:) + smin         *(Jn{idx+1}(4:6,:) - Jn{idx}(4:6,:));
        % Determine effective mass of robot toward obstacle from alpha
        ms = 1/(u.    *Js*invM*Js.   *u);
        % Determine kinetic energy toward obstacle from alpha
        E = 0.5*ms*vu^2;

        % Store this information as well
        DangerInfo.E = E;
        DangerInfo.Js = Js;
        DangerInfo.J = u.      *Js;
        DangerInfo.m = ms;

        allDangerInfo = [allDangerInfo, DangerInfo];
    end
end
```

2) ndCollisionInfo.m:

```
function allCI = findCollisionInfo(pn, Jn, o, ro)
% This function determines which links if any are in contact with the
% obstacle and outputs the information necessary to apply forces to the
% robot at this point and determine contact force constraints.
```

```
%%% Inputs:
% pn: Positions of all frames on the robot, which are the link starting
% points and the end-effector. Each column is a set of (x,y,z) coordinates.
% Jn: Geometric Jacobians of each of these frames.
% o: Position of the obstacles in the world frame. Each column is a set of
% (x,y,z) coordinates.
% ro: Contact radius of obstacle.
%%% Output:
% allCI: array of CI (contact information) structs for the links that are
% determined to be in contact with the obstacle.

N = 6; % Number of links
G = size(o,2); % Number of obstacles
allCI = [];

for m = 1:1:G
    for i = 1:1:N
        % Find minimal distance of link to obstacle and location along
        % link.
        [smin, dmin] = minDistToObst(pn(:,i),pn(:,i+1),o(:,m), [0 1]);
        if dmin < ro
            CI.nl = i; % Link number
            CI.s = smin; % placement along link
            CI.o = o(:,m); % obstacle it is in contact with
            CI.pc = pn(:,i) + smin*(pn(:,i+1) - pn(:,i)); % position of contact point
                in world frame
            CI.poc = CI.pc - CI.o; % Vector from obstacle to contact point.
            CI.doc = dmin; % distance between obstacle and contact point
            CI.uoc = CI.poc/CI.doc; % unit vector from phi to c

            % Determine jacobian of contact point as weighted combination
            % of jacobians for the ends of the link.
            CI.Jc = Jn{i}(4:6,:) + smin*(Jn{i+1}(4:6,:) - Jn{i}(4:6,:));
            CI.invJcT = pinv(CI.Jc.'); % pseudoinverse of transposed jacobian

            allCI = [allCI CI];
        end
    end
end
end
```

3) velToObst.m:

```
function vToO = velToObst(r,v,o)
% This function determines the component of a point velocity in the
% direction of another point.
%%% Input:
% r: The position of the moving point.
% v: The linear velocity of r
% o: The position of the point we want to know the velocity towards.
%%% Output:
% vToO: The velocity of the point r in the direction of the point o.

d = norm(o - r);
u = (o - r)/d;

vToO = u.' * v;
```

```
end
```

4) minDistToObst.m:

```
function [smin, dmin] = minDistToObst(r1, r2, o, s_range)
% This function calculates the minimum distance between a point and a link
% defined by its two ends and a parametrization parameter. By supplying the
% range of the parametrization parameter it is also possible to only
% consider a section of the link.
%%% Input:
% r1: The link's starting point position vector.
% r2: The link's ending point position vector.
% o: The position of the point to which the minimal distance is found.
% s_range: A 2 element array indicating the desired range of the
% parametrization parameter, s. s can be anywhere between 0 and 1. If s = 0
% then the starting point on the link is retrieved from the
% parametrization. If s = 1 then the ending point is returned. Any value in
% between will return the point that is that fraction away from the
% starting point and toward the ending point.
%%% Output:
% smin: The value of s yielding the smallest distance toward o.
% dmin: The smallest distance to o from the specificied section of the
% link.

rs = @(s) r1 + s*(r2 - r1);
d = @(s) norm(o - rs(s));

[smin, dmin] = fminbnd(d, s_range(1), s_range(2));
end
```

5) vuSwitchPoint.m:

```
function sSwitch = vuSwitchPoint(r1, v1, r2, v2, o)
% For a link where one end has a positive velocity toward a point and the
% other end a negative velocity, this function determines where along the
% link the velocity toward the point is zero.
%%% Inputs:
% r1: The position of the link's starting point.
% v1: The velocity of the link's starting point.
% r2: The position of the link's ending point.
% v2: The velocity of the link's ending point.
% o: The position of the point toward which we determine the velocity.
%%% Output:
% sSwitch: The value of the parametrization parameter s along the link for
% which the velocity toward the point o is zero.

syms s
assume(s >=0 & s <= 1);
assume(s, 'real');

rs = r1 + s*(r2-r1);
u = (o - rs)/norm(o - rs);
vs = v1 + s*(v2-v1);
vsu = (u.' * vs);

eqn = vsu == 0;
```

```
sSwitch = double(vpasolve(eqn,s,[0 1]));
end
```

6) checkForCollisions.m:

```
function collision = checkForCollisions(pn, o, ro)
% This function determines if there is contact and is therefore a
% simplified version of findCollisionInfo.m
%%% Inputs:
% pn: Positions of all frames on the robot, which are the link starting
% points and the end-effector.
% o: Position of the obstacle in the world frame.
% ro: Contact radius of obstacle.
%%% Output:
% collision: boolean that is true if contact is detected and false
% otherwise.

collision = false;
N = 6; % Number of link
M = size(o,2); % Number of obstacles

for m = 1:1:M
    for i = 1:1:N
        % Find minimal distance of link to obstacle and location along
        % link.
        [~, dmin] = minDistToObst(pn(:,i),pn(:,i+1),o(:,m), [0 1]);
        if dmin < ro
            collision = true;
            break
        end
    end
end
end
```

C. Kinematics functions

The functions within the Kinematics path are:

1) getJointPoses.m:

```
function jointPoses = getJointPoses(robot, q)
% This function returns the poses of the frames defined on the robot in the
% current coniguration.
%%% Inputs:
% robot: rigidBodyTree model of robot.
% q: Joint position vector
%%% Output:
% jointPoses: cell array containing the poses of the frames. Each pose is a
% 4x4 matrix containing the rotation matrix and position of the frame
% relative to the world frame.

jointPoses = cell(1,7);
for i = 1:1:7
    jointPoses{i} = getTransform(robot, q, robot.BodyNames{i});
end
end
```

2) getJointPositions.m:

```
function jointPoints = getJointPositions(jointPoses)
% This funtion returns the positions of the frames defined in the
% rigidBodyTree model of the robot with respect to the world frame. It can
% be thought of as filtering out the rotation matrices from getJointPoses.
%%% Input:
% jointPoses: A cell array containing the joint poses of the frames on the
% robot.
%%% Output:
% jointPoints: A matrix where each column is a set of (x,y,z) coordinates
% belonging to a frame of the robot.

jointPoints = zeros(3,7);
for i = 1:1:7
    pi = jointPoses{i}(1:3,4);
    jointPoints(:,i) = pi;
end
end
```

3) getJointJacobians.m:

```
function Jn = getJointJacobians(robot,q)
% This function returns the geometric jacobians of the frames defined on
% the robot in the current coniguration.
%%% Inputs:
% robot: rigidBodyTree model of robot.
% q: Joint position vector
%%% Output:
% Jn: cell array containing the geometric jacobians of the frames. Each
% jacobian is a 6xN matrix that allows for the translation of the joint
% velocity vector into the twist of a frame.

Jn = cell(1,7);
for i = 1:1:length(robot.BodyNames)
    Jn{i} = geometricJacobian(robot,q,robot.BodyNames{i});
end
end
```

D. Screw theory functions

The functions within the ScrewTheory path are:

1) Adjoint.m:

```
function AdjH = Adjoint(H)
% This function returns the Adjoint of a given homogeneous pose matrix H.
%%% Input:
% H: The homogeneous matrix expressing the pose of a frame relative to
% another frame.
%%% Output:
% AjdH: The Adjoint of the given homogeneous matrix.

R = H(1:3,1:3); % Get rotation matrix of frame
p_tilde = tilde(H(1:3,4)); % Get tilde form of position of frame

% Define and fill in the Adjoint matrix
AdjH = zeros(6,6);
AdjH(1:3,1:3) = R;
```

```
AdjH(4:6,1:3) = p_tilde*R;
AdjH(4:6,4:6) = R;
end
```

2) as.m:

```
function asA = as(A)
% This function returns the anti-symmetric part of any given matrix.
%%% Input:
% A: Any square matrix
%%% Output:
% asA: anti-symmetric part of A

asA = 0.5*(A-A.');

end
```

3) inverseH.m:

```
function H_j_i = inverseH(H_i_j)
% The function returns the inverse of a given homogeneous transformation or
% pose matrix. That is, if the given matrix expresses the pose of frame i
% with respect to frame j, this function returns the pose of frame j with
% respect to frame i.
%%% Input:
% H_i_j: The pose matrix of a frame i with respect frame j.
%%% Output:
% H_j_i: The pose matrix of frame j with respect to frame i.

H_j_i = eye(4);
H_j_i(1:3,1:3) = H_i_j(1:3,1:3).'; % Transpose the rotation matrix
H_j_i(1:3,4) = -H_i_j(1:3,1:3).'*H_i_j(1:3,4); % Express the location of j in i.
end
```

4) tilde.m:

```
function v_tilde = tilde(v)
% This function returns the tilde form of any twist, angular velocity, or
% position vector. Depending on the length of the input the output is
% either a 3x3 matrix or a 4x4 matrix.
%%% Inputs:
% v: This input vector can have a lenth of either 3 or 6. If it is 3 then
% it is seen as a position or velocity vector. If it is 6 then it is seen
% as a twist vector.
%%% Outputs:
% v_tilde: The tilde form of the input v. If v has length 3 then v_tilde is
% a skew matrix consisting of the components of v. If v has length 6 then
% v_tilde is the tilde form of a twist.

if length(v) == 3
    v_tilde = [0,-v(3),v(2);
               v(3),0,-v(1);
               -v(2),v(1),0];
    return
elseif length(v) == 6
    w_tilde = tilde(v(1:3));
    v_tilde = [w_tilde v(4:6)];
    v_tilde = [v_tilde; zeros(1,4)];
```

```
else
    disp('Input of incorrect length')
    return
end
end
```

E. Robot description

The robot is described within a URDF file, which follows the XML format. The code describing the used model is:

```
<?xml version="1.0"?>
<robot name="myrobot">
    <link name="base">
        <visual>
            <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/>
            <geometry>
                <box size="0.4 0.4 0.2"/>
            </geometry>
        </visual>
    </link>

    <joint name="joint01" type="revolute">
        <axis xyz="0.0 0.0 1.0"/>
        <origin xyz="0.0 0.0 0.1" rpy="0.0 0.0 0.0"/>
        <parent link="base"/>
        <child link="link01"/>
        <limit effort="1000.0" velocity="100.0"/>
    </joint>

    <link name="link01">
        <inertial>
            <origin xyz="0.0 0.0 0.25" rpy="0.0 0.0 0.0"/>
            <mass value="5.0"/>
            <inertia ixx="0.1073" ixy="0.0" ixz="0.0" iyy="0.1073" iyz="0.0"
            izz="0.0063"/>
        </inertial>
        <visual>
            <origin xyz="0.0 0.0 0.25" rpy="0.0 0.0 0.0"/>
            <geometry>
                <cylinder radius="0.05" length="0.5"/>
            </geometry>
        </visual>
    </link>

    <joint name="joint02" type="revolute">
        <axis xyz="1.0 0.0 0.0"/>
        <origin xyz="0.0 0.0 0.5" rpy="0.0 0.0 0.0"/>
        <parent link="link01"/>
        <child link="link02"/>
        <limit effort="1000.0" velocity="100.0"/>
    </joint>

    <link name="link02">
        <inertial>
            <origin xyz="0.0 0.0 0.25" rpy="0.0 0.0 0.0"/>
            <mass value="5.0"/>
            <inertia ixx="0.1073" ixy="0.0" ixz="0.0" iyy="0.1073" iyz="0.0"
```

```
        izz="0.0063" />
    </inertial>
    <visual>
        <origin xyz="0.0 0.0 0.25" rpy="0.0 0.0 0.0" />
        <geometry>
            <cylinder radius="0.05" length="0.5" />
        </geometry>
    </visual>
</link>

<joint name="joint03" type="revolute">
    <axis xyz="0.0 0.0 1.0" />
    <origin xyz="0.0 0.0 0.5" rpy="0.0 0.0 0.0" />
    <parent link="link02" />
    <child link="link03" />
    <limit effort="1000.0" velocity="100.0" />
</joint>

<link name="link03">
    <inertial>
        <origin xyz="0.0 0.0 0.25" rpy="0.0 0.0 0.0" />
        <mass value="5.0" />
        <inertia ixx="0.1073" ixy="0.0" ixz="0.0" iyy="0.1073" iyz="0.0"
        izz="0.0063" />
    </inertial>
    <visual>
        <origin xyz="0.0 0.0 0.25" rpy="0.0 0.0 0.0" />
        <geometry>
            <cylinder radius="0.05" length="0.5" />
        </geometry>
    </visual>
</link>

<joint name="joint04" type="revolute">
    <axis xyz="1.0 0.0 0.0" />
    <origin xyz="0.0 0.0 0.5" rpy="0.0 0.0 0.0" />
    <parent link="link03" />
    <child link="link04" />
    <limit effort="1000.0" velocity="100.0" />
</joint>

<link name="link04">
    <inertial>
        <origin xyz="0.0 0.0 0.25" rpy="0.0 0.0 0.0" />
        <mass value="5.0" />
        <inertia ixx="0.1073" ixy="0.0" ixz="0.0" iyy="0.1073" iyz="0.0"
        izz="0.0063" />
    </inertial>
    <visual>
        <origin xyz="0.0 0.0 0.25" rpy="0.0 0.0 0.0" />
        <geometry>
            <cylinder radius="0.05" length="0.5" />
        </geometry>
    </visual>
</link>
```

```
<joint name="joint05" type="revolute">
    <origin xyz="0.0 0.0 0.5" rpy="0.0 0.0 0.0"/>
    <parent link="link04"/>
    <child link="link05"/>
    <axis xyz="0.0 0.0 1.0"/>
    <limit effort="1000.0" velocity="100.0"/>
</joint>

<link name="link05">
    <inertial>
        <origin xyz="0.0 0.0 0.25" rpy="0.0 0.0 0.0"/>
        <mass value="5.0"/>
        <inertia ixx="0.1073" ixy="0.0" ixz="0.0" iyy="0.1073" iyz="0.0"
        izz="0.0063"/>
    </inertial>
    <visual>
        <origin xyz="0.0 0.0 0.25" rpy="0.0 0.0 0.0"/>
        <geometry>
            <cylinder radius="0.05" length="0.5"/>
        </geometry>
    </visual>
</link>

<joint name="joint06" type="revolute">
    <origin xyz="0.0 0.0 0.5" rpy="0.0 0.0 0.0"/>
    <parent link="link05"/>
    <child link="link06"/>
    <axis xyz="1.0 0.0 0.0"/>
    <limit effort="1000.0" velocity="100.0"/>
</joint>

<link name="link06">
    <inertial>
        <origin xyz="0.0 0.0 0.25" rpy="0.0 0.0 0.0"/>
        <mass value="5.0"/>
        <inertia ixx="0.1073" ixy="0.0" ixz="0.0" iyy="0.1073" iyz="0.0"
        izz="0.0063"/>
    </inertial>
    <visual>
        <origin xyz="0.0 0.0 0.25" rpy="0.0 0.0 0.0"/>
        <geometry>
            <cylinder radius="0.05" length="0.5"/>
        </geometry>
    </visual>
</link>

<joint name="jointEE" type="fixed">
    <origin xyz="0.0 0.0 0.5" rpy="0.0 0.0 0.0"/>
    <parent link="link06"/>
    <child link="endeffector"/>
    <axis xyz="0.0 0.0 0.0"/>
    <limit lower="0.0" upper="0.0" effort="0.0" velocity="0.0"/>
</joint>

<link name="endeffector">
```

```
    </link>
</robot>
```