Implementing machine learning in industrial robots for better human-robot cooperation

René Heijdens, S1424378

August 24, 2017

Abstract

Industrial robotic control has not changed significantly over the last years. Robots still rely on predetermined machine instructions in order to work. Newly developed cooperative robots uses the same control mechanisms. This method has little input from the environment to make decisions. This obstructs human-robotic cooperation during operation. In order to improve human-robotic cooperation, a design of a new controller for industrial robots is presented enabling a more human-friendly cooperation trough machine learning algorithms. These algorithms create other methods to program industrial robots, and makes robots more flexible. A prototype is build as a proof of concept. It also forms a base which enables future development of software for the ABB YuMi robot.

1 Introduction

Industrial robots (IR) are designed for automation in manufacturing processes. They are able to reproduce movements with high accuracy, repeatability and efficiency to improve the process. These kinds of robots are mainly designed to work in a closed and controlled environment, a so-called work cell, to ensure maximum productivity and to prevent interference with the current task.

New technologies are being developed for more efficient and flexible robotic applications. Robots that can be set-up with minimal efforts can be used for low-volume and high variant applications. This can increase the throughput and quality of the production. [16]

Large IR-manufactures also produce robots that can work outside cages and in cooperation with humans. These so-called collaborative robots already have the sufficient safety mechanisms for save operation. One major drawback with these collaborative robots is that they still require some sort of programming to function. Once programmed, the robot cannot make decisions based on its environment due to the lack of sensors. Current IR's are equipped with sensors to ensure accurate movements but lack the sensors to sense the environment. If robots need to become more collaborative with workers, they need to detect the environment and make situation dependence responses.

In order to make a robot more collaborative, it requires two things. First, it has to detect what to do and learn how to deal with human interaction. Second, a computer with additional sensors must control the robot in real-time, while maintaining their safety, efficiency, robustness and reliability. Once these two aspects merge, it creates the ideal collaborative robot.

This paper is about the design of a controller which is able to do the two things mentioned above. Different aspects of the controller will be discussed. These aspects are divided into Human-Robotic interaction, integration of machine learning algorithms in the controller and the current prototype. The major reason of this prototype is to prove that current Industrial Robots can indeed be controlled by custom software.

2 Human-Robotic Interaction

Current research on human-robotic interaction (HRI) have lead to movement restriction controllers, like ABB smart move, and collaborative robots like the RethinkRobotic Baxter and ABB YuMi. While these robots do have promising properties for human interaction, they are focussed on completely new robotic systems, excluding the possibility of retrofitting the current robots with new HRI controllers. Human-robotic interaction needs to add value to the process in order to succeed. This means that the advantages of both the human and robot are used within the same process. The quality of the process increases due to better operation steps. Robots can then be integrated into flexible environments which need additional accuracy. The accuracy can be delivered by the robot while maintaining the flexibility humans offer. If any person can

work with robots, it opens up possibilities for robots to be used outside controlled environments. People do not require training to work with robots, so it can be used in applications where lots of different people have to work together with the robot. The first machines could only be operated by experts, but are now available for many. It is now for the robot to undergo the same development.

2.1 Safety

One major aspect of the design of this controller is safety. Safety can be divided into two different aspects; physical safety and psychological safety. Physical safety is the most straight forward. It provides unwanted physical contact which may harm the human. Psychological safety ensures that the robot does not cause additional stress and discomfort to the worker. Fast-moving or sharp parts may discomfort the worker. While the physical safety ensures that the worker is safe during operation, a robot can still cause a lot of stress and is, therefore, psychological unsafe [18]. By adapting secondary aspects of the operation, like taken path or speed, workers can feel more comfortable during cooperation.

2.2 Physical Safety

During action, the controller can predict which path the robot takes and whether this path is free of any obstacles. A collision with any object is unwanted unless it is been trained by the action classifier. Wanted collisions are for instance collisions of different parts with a tool. Although it is still a wanted collision, speed decreases during the approach of objects, to prevent damage. If the current taken path gets interrupted by an object, like hands of a worker or unknown objects, the controller makes changes for the taken path of the robot to avoid the hands, but continues its current task, providing the physical safety barrier.

3 Controller Aspects

The controller has to control multiple aspects of the robot by using the input of the user. These aspects can be divided into two categories.

1. Technical operation
2. Social operation

This division is made because of the addition of the cooperation with humans, which adds the social aspect to the robotic controller instead of purely technical operation.

3.1 Technical operation

This part focusses on the skill of the robot. The controller needs to learn the basics actions necessary to fulfil tasks. The controller learns a new skill on how to approach or pick up objects and how to manipulate input into desired output. Also, the intention of the tasks has to be clear to the robot. This is essential for the controller to execute the right tasks to produce the desired output [13]. If, for instance, the desired output is wrong or unknown, the feedback produced is based on wrong assumptions by the robot. This generates false feedback and degrades the algorithm. This feedback will be discussed in paragraph 4.7. Current operation of a robot have instructions made ahead of time. The difference between the current operation and this technical operation is that due to the influence of the social operation, the technical operation changes. This means that the instructions have to be generated in real time.

3.2 Social operation

Social operation focusses on how the robot accomplishes the task. Social operation requires human behaviour inputs. It makes decisions according to the mood of the worker, which is discussed in paragraph 3.3. The controller adapts its behaviour according to the mood of the worker. This influences the robot's behaviour in secondary aspects. These aspects do not change the basic tasks, but it changes the handling of these tasks [18]. In order for humans to work together with robotics, there are several aspects that need to be reconsidered for cooperation. These aspects are divided into four categories; Anticipate, Transparency, customization and right objectives [23].

3.2.1 Anticipate

In human-human cooperation, people coordinate and communicate their behaviour through physical and social-cognitive levels. Humans anticipate on the others' movement trough these communication channels [20]. For instance, during normal human-human cooperation, handling over tools, like passing a wrench, is a basic action, which highly relies on these communication levels. During the handover process, several actions are involved for passing over equipment, these are shown in figure 1.

Robots need to be able to communicate to the worker trough these two communication levels as well in order for a good cooperation.

3.2.2 Transparency

Communication goes both ways. People need to anticipate on the robots' behaviour as well. People make a lot of assumptions when they watch other people, for instance, information about a part which they carry. If a robot hands over a certain part, the worker has to make an assumption about the weight of the part, but also when to accept the handover.[23, 20]

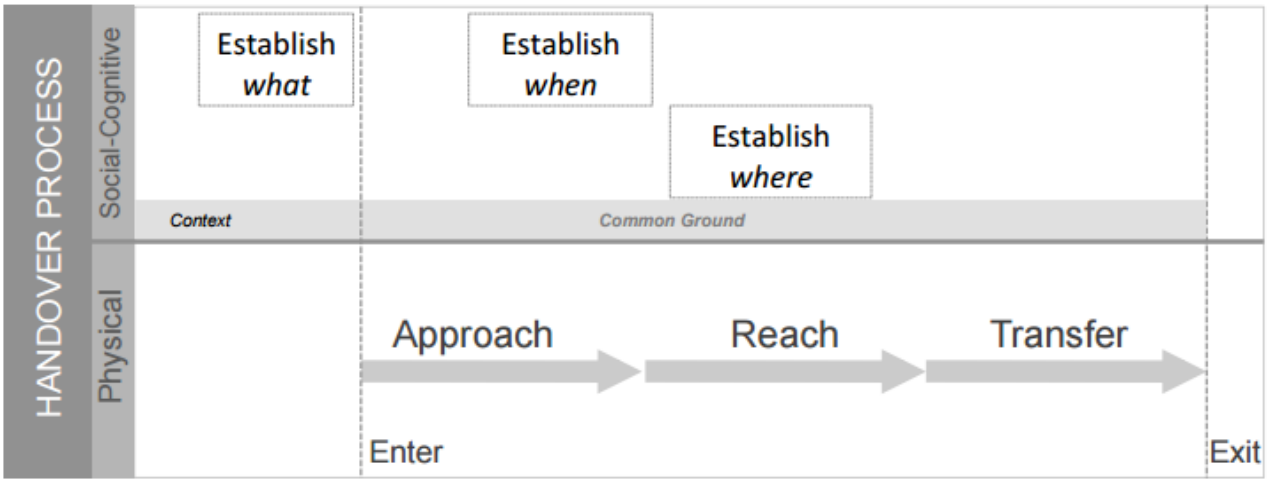

Figure 1: The canonical handover process [20]

3.2.3 Customization

During the design of self-driving cars, tests about which driving style the car should have were being performed. This test was about the connection between desired driving style of the car, in relation to the driving style of the respondent. The test indicated that people prefer a different driving style than their own, even though they thought they had the preferred driving style as their own. This shows that it is hard to predict what the worker prefers as the driving style because the preferred driving style did not match their own driving style [11]. This test shows that one general solution does not fit all of the workers. This shows that it is important to get feedback from the individual during operation rather than provide a general way of working for each worker.

3.2.4 Right objectives

In the current situation, an ideal movement path could easily be calculated by, for example, the shortest path, fastest path or the path which minimises forces on the robot. When robots work with humans, this assumption does not work. For instance, how do workers respond on the acceleration or velocity of a robot? What is an ideal path for humans? This requires emotional input in the robotic controller in order to deal with each individual [18, 12]. This way, the robot can determine by itself which taken path or speed was considered the best for the worker. So during the operation, the robot learns what the right objectives are

3.3 Social Input

The state of mind of the user has to be linked to a variable in order to make changes to the behaviour of the robot. A workers' state of mind can be influenced by many aspects. Therefore it is of high importance to measure the change. If a worker arrived stressed, it may be caused by an external factor than by the robot.

3.3.1 Stress indicator

The input for the controller for this social aspect is the stress level of the worker. Stress can be detected by sensors placed on the workers body. Tests have been done by placing Galvanic Skin Response (GSR) sensors on the workers' body to read the electro dermal response, which is an anxiety indicator [19]. If the movements cause high GSR repose, it indicates a stressful movement by the robot [18]. Other physiological responses to stress is the cardiac response [19]. Sensors to measure heartbeat are nowadays available as consumer electronics, like fitness bands. These fitness bands measure heart rate accurate [22], which enables them to be used for this application. There are multiple aspects which can influence these variables. Working can be a physical intensive task, which may influence these measurements. During physical labour, the heart rate and sweat production increases as well. It is, therefore, important that these measurements take place during the movement of the robot, to eliminate the influence of the effects of physical labour.

These variables combined compose one variable called the stress indicator. However, not all of these aspects have the same influence on the stress indicator. This is managed by the algorithm by means of normalising, which will be discussed in paragraph 4.2.

This stress indicator should provide the psychological safety for the worker to feel comfortable with the robot. Physical safety is easier to program during training, but the knowledge about the influence of certain actions on the workers' feeling is not integrated during the first use of the robot. Human behaviour is hard to predict in models, so this is only trainable during operation. Therefore, during the start of implementation, it is necessary to keep in mind that the robot might make some manoeuvres in which the worker does not feel completely comfortable. Due to the physical safety, the workers are not in danger.

3.4 Robotic Control

Current control of Robots relies on a pre-determined machine code. These codes are generated by software that simulates the manufacturing process. This generates a set of instructions, resulting in a highly optimised but inflexible process. Robots cannot deal with situation dependent responses. In case of abnormal situations, like interrupts, the robot stops as a safety precaution. For humans to interact with robots, there have to be other ways of providing input to the robot, eliminating the need for the pre-determined machine code. The robot has to respond to the current situation. This requires a different approach on providing instructions to the robot and make adjustments to the instructions if necessary. In the following paragraphs, machine learning is introduced as a way to generate instructions for the robot in real time, and eliminate the need of manually writing instructions.

3.4.1 Machine Learning

The desired controller uses machine learning algorithms to create the program. The difference between traditional programming and machine learning is the way the program is generated. While in traditional programming, the developer delivers the data and program code, in machine learning, the output and data are delivered to the algorithm which consequently produces the program, as shown in figure 2. [7]

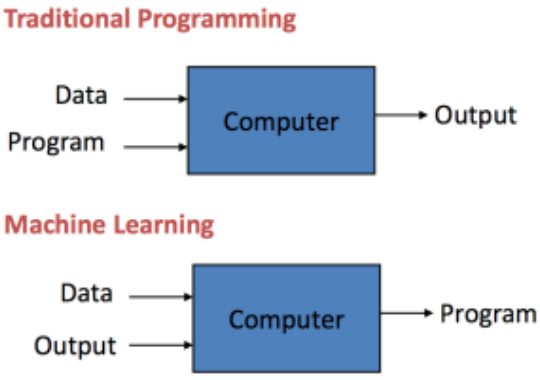

Figure 2: Traditional programming and machine learning [7]

3.5 Differences in machine learning algorithms

There are two different categories of machine learning algorithms. Regression and classification algorithms. Regression outputs continuous values which for which $x \in \mathbb{R}$. Classification algorithms generate discrete outputs. This means that the output is a set of predetermined elements, for instance, a for set of possible outputs A, $x \in A$.

A classification algorithm returns a list of all the states in the set with the possibility of that state. The controller requires classifying different environment states and objects. Therefore, a classification algorithm will be used for this controller. Current algorithms do support multi-class classification. This means that the algorithm detects states and objects at the same time, making it possible to detect several different objects and states at the same time. One useful feature of classification algorithms is that it can be retrained without losing previous data. If a robot learns a new skill, it does not forget the previous skills it acquired[8]. The newly trained dataset is added to the current set of outputs. So if two of the same algorithms, but with different sets (set A and set B), can be merged, resulting in an output $x \in (A + B)$.

This opens up new possibilities for robots to be used in more flexible environments. Datasets of multiple robots with the same algorithm can be merged to create a universal controller for multiple tasks and multiple robots. This means that it is not necessary to reprogram the robot for other tasks, but it recognises the tasks which need to be performed by itself.

This can be really useful for flexible assembly robots. For example, the algorithm recognises the parts given to the robot and can act accordingly, without explicitly telling the robot what to do with it.

4 Controller Design

The previous paragraphs described a set of characteristics and functionalities for the controller. This paragraph will describe how these characteristics and functionalities will be implemented in machine learning algorithms. While there are already a lot of different classifications algorithms, there is not a general algorithm which performs the best in every situation. Different algorithms do work differently on mathematical levels, the general workflow is the same for all the algorithms. Figure 3 shows a general workflow of machine learning algorithms. [15] This workflow will be used throughout this paragraph. Each title of a paragraph refers to a state in this workflow.

Figure 3 is a form of an agent-environment loop, also known as an intelligent agent, which is an autonomous algorithm which improves itself by observing trough sensors and evaluate after each action, as shown in figure 4

Figure 4: agent-environment loop [5]

In the last paragraph, a selection of appropriate algorithms is discussed.

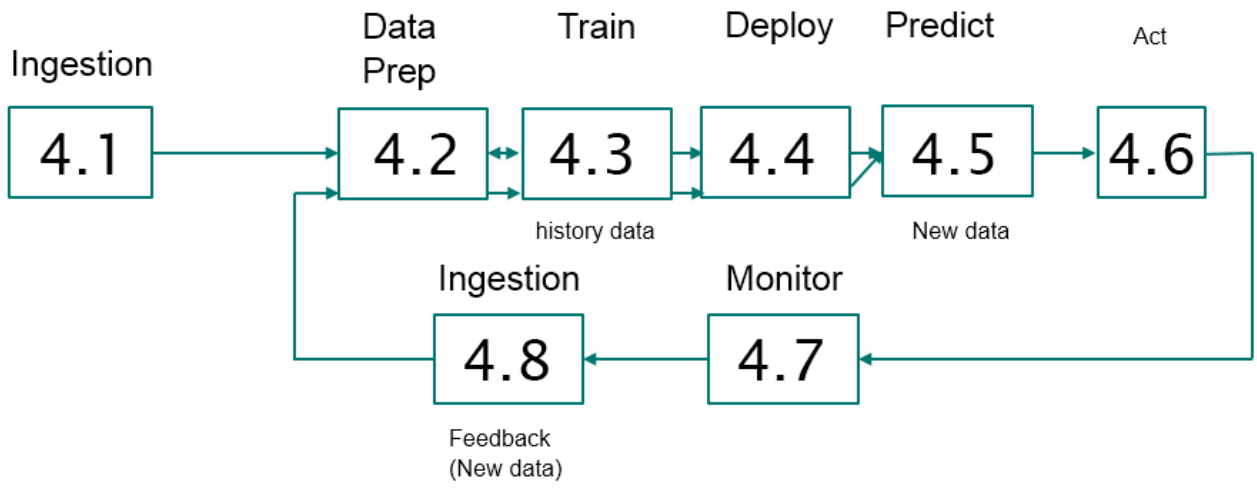

Figure 3: Machine Learning Workflow [15]

4.1 Ingestion

Ingestion is the absorption of information by the algorithm. It is the input of the controller. The input consists of several observations. Based on these observations, the algorithm makes a prediction based on the preparation data and the ingestion. The input consists of several parts in the robotic controller.

- Camera images
- Depth images
- Stress indicator
- Object sizes
- Object coordinates

This data forms the basis on which the algorithm makes its decision.

4.2 Prep data

During this stage, the input gets modified by the algorithm. The representation of the data changes, making it able to distinguish different aspects of the input based on a unique representation. This representation is later used to classify different objects. Modification of data can be normalizing input variables for equal influence on the algorithm or image modifications like convolutional functions.

Training data is separated into three datasets; training, testing and validation. These sets are divided so the algorithms' accuracy can be evaluated with the preparation data alone. At the start of the training, the set gets divided into two sets, training + test set and the validation set. These sets are divided so that the trained weights can be evaluated during training. The first set, training + test is divided for each learning step, which means that an image can be used for training and for weight evaluation at different training steps. The validation set is also used to analyse the trained weights. The difference is that this set is never used for training purposes. It can monitors algorithm is overfitted or underfitted. This results in two different accuracy values. From these values, three situations can occur;

- Both accuracy values are low.

  This represents a underfitted algorithm or high-bias algorithm. This may indicate that the dataset does not represent the desired classification, the dataset is not large enough or the dataset contains false data.

- Both values indicate a high accuracy.

  This indicates a well-trained algorithm.

- Validation accuracy is high, while test accuracy is low.

  The algorithm fits the training data too well, while it still cannot produce reliable outputs. This indicates a overfitted algorithm, also known as a high-variance algorithm. This is a result of an algorithm which is too much focussed on the training set and ignoring the important aspects of the input. The algorithm might be focussed on the noise in the training set rather than the important aspects. For instance a function with input vector X and integer p and a output matrix $[X^1, X^2, .., X^p]$ is prone for being high-variance. Using too high polynomial functions might wrongfully exaggerate the influence of certain values on the output.

Figure 5 shows a graphical explanation of these characteristics.

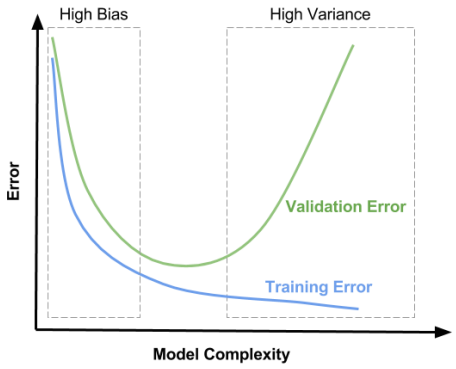

Figure 5: Results during training. The vertical axes is the error between the predicted item compared to the actual item. The horizontal axes is the complexity of the total algorithm, for example higher polynomial equations, more layers in a neural network or more perceptrons per layer. [2]

4.3 Train

During training of the algorithm, the presence metadata about the learning data enables different learning styles. The data, as described in paragraph 4.2 is already being identified. This means that the metadata categorises the training data to enable supervised learning. Supervised learning is better suited for this application in comparison to semi-supervised and non-supervised learning. The algorithm does not need to recognise objects which are not included in the training data. In case of non-supervised learning, the algorithm learns the relation between the images in the training data itself. Semi-supervised learning is a combination of supervised a non-supervised learning.

Due to the division of preparation data, training of the complete algorithm is divided into two aspects. First, all of the parts and tools need to be learned in an image classifier network. The static images are placed into a specific folder, which functions as the input of the algorithm.

A few variables are needed for the model to be trained. These variables prevent a high-bias or high-variance algorithm if set-up properly.

1. Training Steps

   This implies the total amount of training the algorithm needs in order to train the weights.

2. Distortions

   Distortions will make minor changes to the training data to increase the number of images. It improves the results by randomly deforming, cropping, or brightening the training images. This improves the algorithm how to deal with distorted images which occur in the real world.

3. Learning rate

   This controls the influence of one single image in a complete algorithm. High learning rate can increase the error rate, causing a less accurate algorithm. A smaller learning rate causes slower learning of the algorithm.

4. Train batch size

   The number of images that will be loaded during one training round. More images will cause longer time per step but also increases the accuracy. High train batch size prevents an algorithm for overfitting.

During training, the accuracy values of the test and validation set can be monitored. If the values indicate a high-bias or high-variance algorithm, interruption of the training is necessary to prevent unwanted weights. Carefully choosing these variables can result in higher accurate algorithms.

4.4 Deploy

If an algorithm is deployed, the input data gets transformed and modified using the previously determined weights. These steps are build to distinguish several unique features into representative numbers which will be loaded into the classifier.

4.5 Predict

An algorithm predicts the output by a classifier, also known as an activation function. A classifier is the last step of the algorithm, linking the previously processed data to a probability of an object. Classifier functions can be the sigmoid function, ReLU function or a logistic function.

The classifier requires several items to detect. The robot needs to be able to perform several complex tasks. To simplify this process, object can be divided into three categories;

- Parts

  The objects which are processed by the robot. Also, if multiple parts are processed by the robot, like assemblies, it creates a new part and loses the previous parts.

- Tools

  Necessary objects to perform a certain operation. These tools are still available after manipulation.

- Actions

  The robotic actions needed to perform the operation.

How this division of objects will be handled in the data-flow will be discussed in the next two paragraphs.

4.5.1 Parts and tools classifier

The parts and tools training data can be generated by computer generated renders. The assumption is made that the objects handled by the robot are modelled using 3D modelling software. This enables the possibility to create renders from the parts in all possible directions. These renders contain only images of the parts itself. The images have to represent the images during regular operation. The images from the robot contains an object mask, which only displays the object with a black background, as shown in appendix 13. This prevents the algorithm from focussing on unrelated objects. It also prevents unintended learning. Unintended learning is the phenomenon that the algorithm focusses on unrelated objects. This process can be automated to generate sufficient data in order for accurate recognition results.

4.5.2 Action classifier

The action classifier consists of two parts. First, it uses the parts and tools classifier to make an inventory of the needed objects for each action. This inventory helps the algorithm to decide whether to start an action and what kind of action. The second part is the action itself. Using visual object tracking, the position of non-pre-trained objects can be followed [17]. These algorithms detect several objects and trace the followed path. This is also known as imitation learning or behavioural cloning. It learns actions by observation and demonstration. An observation is the view of the current status, while a demonstration is a part of a task which is being performed. This means that one task consists of multiple demonstrations.

The algorithm also needs to be able to make decisions in a sequence based on the input of the object classifier and current observations. Any algorithm for learning in decision-making problems is suitable for this task. A suitable algorithm is described in article [13], which uses multiple machine learning algorithm to identify different tasks. It described a task of block stacking, which a variable amount of blocks to stack. This demo is a proof in which an algorithm is able to pick up certain objects and align them relative to other objects.

4.6 Act

By dividing the objects between these criteria, it creates an understanding of different kind of objects, and whether it can start the actions.

For instance, if the algorithm detects a certain part, it then can check if all the parts and tools are available. If not, it can actively ask the worker for the missing part(s) or tool(s). It merges information from the image classifier, together with object sizes and coordinates from the computer vision, to make an inventory of the current situation.

Once an inventory matches that of the required inventory of a certain action, the action can be started.

4.7 Monitor

The feedback loop which enables the algorithm to improve itself during operation. The feedback is provided by several aspects.

1. Observation or object

   An environment-specific object, in this case, a part or tool.

2. Reward

   The value given to the previous action of the algorithm. The algorithms' goal is to increase the reward. The reward can be composed out of several variables. These variables are the stress level, time needed and quality of the output.

3. Done

   A moment when the environment gets reset again. Most tasks have a clear start and end position.

4. Info

   Diagnostic information for debugging. This part is primarily useful for debugging and learning the algorithm.

This learning is called online learning. This means that not the whole data is loaded, but it is trained by giving updates one by one, which calculates the error produced by the output of the algorithm. This updates the weight parameters after each instance, which improves it after each instance.

4.8 Feedback Ingestion

The results from the monitor part are linked to the previous action by the robot to indicate whether the action created a desired output. This data can also be saved to store a data set for the development of other algorithms.

This is in line in how new employees are being incorporated into new jobs. New employees have to learn tasks by watching other employees do the job. This way, new employees generates knowledge about the task and how to reproduce it.

4.9 Machine learning network

The previous paragraphs described in general how machine learning algorithms can be used for an application like this. The next two paragraphs describe in more detail some state of the art classification networks which can be used for the controller.

4.9.1 Convolutional neural networks

Current research has proven that deep convolutional neural networks (CNN) are good at recognising different objects. Several algorithms have been developed over time improving the accuracy and efficiency of recognising objects. Deep convolutional neural networks are based on the workings of the brain. Such networks contain multiple processing units called perceptron. Each perceptron has a connection weight w_j and output y, which applies a certain mathematical equation to the input. A simple example of a certain perceptron is the weighted sum of the inputs, see equation 1.

$$y = \sum_{j=1}^{d} w_j x_j + w_0 \tag{1}$$

w_0 makes the model more general. It is a bias unit, which always has the value of 1. During training, the value weight w is learned, so that given an input x, output y is generated. The error is calculated by the difference between the desired output and the generated output. [10]

The input x is a mathematical representation of an object. For instance, '7', '0111' and 'VII' are all representations of the number seven, but there is a difference in algorithm if these representations are added to each other. The same goes for image recognition. There are several ways to represent an image. Image recognition works with arrays that contain the values of each pixel. It is of high importance that the input of the algorithm is constant throughout the whole training and operation. This is to prevent false readouts due to wrong representations of the data. Image recognition works with positives and negatives weights. This means that for some places in the array, some values are desirable or undesirable for a specific place, causing a positive or negative weight w [10, 8].

For example, handwriting recognition can be recognised easily because it is either black or white at a specific place. The desired (Blue) and undesired (Red) places for black values are shown in figure 6.

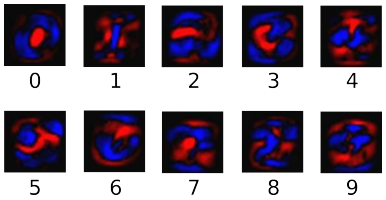

Figure 6: positive and negative weights [8]

The challenge hereby is to create a network with a low error rate, by linking perceptrons to each other with each a specific mathematical equation so that the network is able to detect unique values for each input, see figure 7 for an example. These networks are also known as Multilayer perceptron networks (MLP).

The addition of multiple datasets is also known as transfer learning or retraining. This technique adds specific classifiers to the last layer of the network, which links output from the previous layers to objects. This is the last step in identifying objects in images.

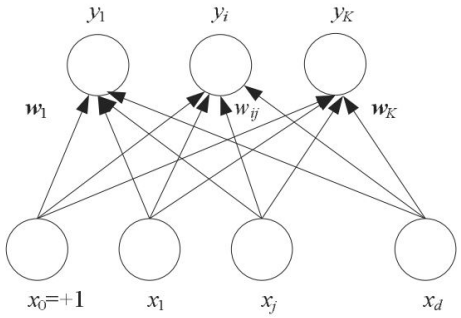

Figure 7: K parallel perceptrons. X_j, $j = 0, ..., d$ as inputs. $y_i, i = 1, ..., k$ as outputs. w_1, w_{ij}, w_k are the weights. Each output is the weighted sum of the inputs. [10]

The unique feature of this type of network is that it uses the convolutional equation to merge different layers of an image. The input image is a 3D image, with the x and y pixels as the first 2 dimension, while the RGB colours represent the depth. Using the convolutional equation, the image gets distorted in an image with smaller x and y, but increased depth. This creates lots of data. Using mean or max pooling, the data is reduced, while maintaining the information on the image. An overview of a convolutional neural network is shown in figure 8.

This classifier returns the type of object. Combining this with the measurements from computer vision, that is going to be discussed in paragraph 6.1, it is able to precisely identify different kinds of objects. As an example, the image classifier recognises an object as a bolt. Computer vision is able to measure the size of the object, providing a complete understanding of the bolt including its size. The lowest error rate by a CNN is currently 21.2%. This is achieved by a network called Inception V3 [21]. This indicates that a CNN alone is not enough for sufficient image recognition. Research on these networks is still ongoing. The accuracy will improve over time.

For training purposes only, datasets from other sources can be used [1, 3]. This is primarily testing the algorithm, other than improving the recognition of parts and tools.

4.9.2 Long short term memory

For imitation learning, it is important that some information gets stored inside the algorithm for a period. In regular neural networks, the data is only used once in a perceptron. This sort of neural network is called a feed-forward neural network. This means there are no loops inside the network itself, indicating the information flow goes one direction. Imitation learning networks contain therefore perceptrons which can

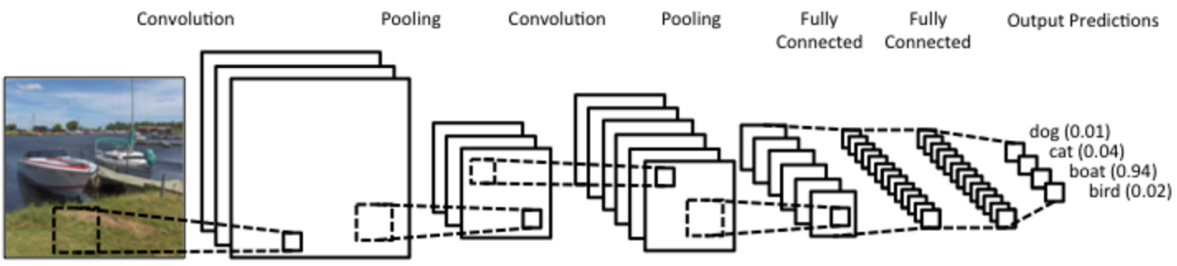

Figure 8: Schematic overview of a CNN [6]

store information. This can be achieved by building loops inside perceptrons. The value stays the same inside a perceptron during operation. The perceptron stays closed and does not interfere with other perceptrons. The perceptron can be read whenever needed while maintaining the value. The perceptron can also be overwritten if the value needs to be updated. This is a way of introducing a simple memory cell inside a network for storing information about previous actions for a short time.

These networks are used for imitation learning because it needs to store some information about the past for a short time in order to finish their behaviour. The difference of these memory cells compared to the weights of perceptrons, is that this can be overwritten at any time. It is also not related to the training data and therefore considered as short term memory.

4.10 Conclusion

While the function and implementation of machine learning algorithms in the controller is clear, there are still a lot of details that need to be examined. The type of algorithm and the exact implementation is still unclear and requires more research to implement these networks in a controller.

5 Integration of the controller to current IR's

In order for a custom software to work with robots, computers need to control the robot in real time and know the current status of the robot. This opens the opportunity to add different sensors to the computer to sense the environment. Current industrial robots have little opportunities to make changes to the software. The regular software is published as closed source software, which leaves little possibility to make changes to the software. Robot manufacturers do not make their software compatible with other brands. Information about the software itself is scarce. This makes it hard to control the robot with software other than provided by the manufacturer itself.[14]

Robotic manufacturer ABB published an open source driver to control ABB robots trough a TCP socket interface [9]. This driver enables programmers to control the robot using simple commands. It is set up in such a way that the robot does not need any modifications to the software itself. The driver consists of two different programs which suit the programming language of the targeted device. Even though this standard ABB driver has little functionality, developers created their own forks based on this driver. One of these forks is Robo.op [14]. This software is developed for artists to use industrial robots and to avoid the limitations set up by the manufacturers. It is designed in such a way that custom tools can be used together with robots, share tools and knowledge across different robot platforms and bypass the expensive tools and software offered by robotic manufacturers.

5.1 Certification

Dutch labour law requires safe and certified machinery in working areas. For manufacturers to use these human-robotic controllers, it needs to be certified to be allowed in production environment. Current collaborative robots do not need additional certification because the current safety features of those robots remain enabled. To use current industrial robots for collaborative work, the safety features have to be recertification in order to be used in production environment.

6 Prototype

The prototype is build as a proof of concept. It based on the ABB YuMi robot. This robot is the first cooperative robot build by ABB, specially designed to operate next to humans. This was the only robot available, and therefore the only available option. The current prototype is a full robot controller on a regular PC without modifying the operating system of the YuMi. It has been build so that it can be extended with machine learning algorithms. Currently, it only recognises images from computer vision, without any added intelligence. The controller sends instructions and receives information. A schematic diagram about the current prototype and the desired controller is shown in appendix 9 and 10.

6.1 Environment input

The robot used has almost no input from the environment. It requires input sources to make a 3D map of the environment. This requires X, Y and Z inputs. For this input, a Kinect camera is used. This camera contains several cameras including an RGB camera and Depth camera. The Kinect camera is mounted on top of the robot to have a clear view of the working area, see appendix 25.

6.1.1 Depth camera processing

The primary source of input is the depth camera. The depth camera maps a depth value in a plane. This creates a 3D image of X and Y in pixels and pixel values in millimetres and is able to work under any light condition.

During initialization, the depth camera scans the environment and saves the image. This image is the reference image to detect changes in the environment. The input stream from the depth camera is compared to the reference image. lower pixel values are stored as 1's. Larger or equal values are stored as 0's. The result is in appendix 12.

It may cause some problems once multiple objects are stacked on top of each other, but in case of stacked objects, the depth layer can be processed differently to create a masking layer suited for a stacked application. Instead of just comparing with the reference image, values outside the given range can be blacked out as well. It creates an image at a specific height.

6.1.2 Object RGB Image

The masking image contains the extruded geometry of the objects. This image is created for two reasons. First, the binary image is used as a masking layer for the RGB image. The black regions on the binary image will be blacked out on the RGB image. This creates an image in which only the objects are shown, which is perfect for image recognition, see appendix 13. From this layer, a cutout can be created containing a single object, providing a uniform input for a machine learning algorithm.

6.1.3 Basic shape detection

The second reason for this image is basic shape detection. Basic shape detection detects squares, triangles and circles. All of the appearances of the items are filtered, leaving out any possible detection of prints on the parts.

The basic shape detection algorithm uses several functions from a computer vision library. First, it uses an edge detection function to detect the outer shape of the white objects, see appendix 14. This image is then transferred to a line detection function and contour detection function. The line detection function returns an array of lines, Which consist of an end and a start. The contour detection function returns an object VectorOfPoints which describes a shape in several vectors connected to points. Objects with 3 vectors are considered as a triangle. Rectangles have 4 vectors at around 90°. A rotated rectangle is created from the rectangle contour. A rotated rectangle is a bounding box which fits the contour the best, in other words, a rectangle with the smallest surface. This rectangle contains a height, width, centre X coordinate, centre Y coordinate in pixels and clockwise rotation in Euler degrees on the horizontal axe.

These basic shapes are drawn into an image for debugging purposes, shown in appendix 15. It indicates which objects are recognised. Only the object properties from a rotated rectangle are necessary for the controller.

6.1.4 Real world units

The computer vision object units need to be transformed to real world units.

So set the rotation of a specific object, it needs the angles set by quaternion rotational units. These units describe a rotation of an object in a 3D plane. The rotated rectangle describes the rotation of the rectangle in Euler angles on a single plane. ABB does support Euler angles, but only for offset moving, which uses the planes of the current tool as reference plane to rotate. The robot needs the absolute rotation of the object in quaternion units to approach new objects. This conversion can be done using sine and cosine functions shown below.

$$q = \begin{bmatrix} cos(\theta/2)cos(\phi/2)cos(\psi/2) + sin(\theta/2)sin(\phi/2)sin(\psi/2) \\ sin(\theta/2)cos(\phi/2)cos(\psi/2) - cos(\theta/2)sin(\phi/2)sin(\psi/2) \\ cos(\theta/2)sine(\phi/2)cos(\psi/2) + sin(\theta/2)cos(\phi/2)sin(\psi/2) \\ cos(\theta/2)cos(\phi/2)sine(\psi/2) - sin(\theta/2)sin(\phi/2)cos(\psi/2) \end{bmatrix}$$

The code in c# is shown in appendix 27

6.1.5 Camera Calibration

The robot uses the X, Y, Z coordinate system. The robot defines the origin of the robot in the centre of the robots' mount. The coordinate system of the camera has to be linked to the robots' coordinate system.

During camera calibration, the camera detects two coloured dots. The rest of the image is filtered using an HSV image, as shown in appendix figure 17, 18 and 19. HSV stands for hue, saturation and value. the hue and saturation value does not change in different lighting conditions, making it ideal for threshold filtering, as shown in appendix figure 21. The position of these dots in relation to the robots' origin is given to the algorithm. From these dots, a pixel to millimetre variable is calculated, together with the height of the camera. These variables are later used to measure the size of an object. By using the pixel size and height in millimetres, the ratio between pixels and millimetres can be calculated. If these dots are mounted on the robot arm, the calibration can be even more precise.

The current robot position can be linked to the position of the dots. Hereby, the relative position of the dots with the robot coordinates is constant.

For the angle in which the camera has been placed, a different calculation is used. The table, on which the robot is mounted, has a flat surface. By readings from the depth camera, the camera angle is calculated. The readings are shown in appendix 23 and 24. The angle of the camera can be calculated from the depth readings of two spots.

Lens correction is applied for a linear camera image. This eliminated radial image distortions. Already existing camera calibration algorithm can calculate lens distortion over the whole image. This variable can later be used to distort the image from the camera for linear calculation [4]. This is a constant value, which means this has to be done only once.

6.2 Robotic communication

The software used to control the ABB YuMi robot is based on the Robo.op software. Modifications are made to make it work with the YuMi and Kinect. The RAPID code is been extended to add the extra functionalities the YuMi has to offer. It is rewritten so that new functionalities introduced by ABB can be implemented in this code.

The computer software is based on the Java code from Robo.op, which is rewritten into c#. The library contains a class robot with non-static variables. New classes can be made, which accesses different robots at different IP addresses and ports. Multiple robots can be controlled at the same time by the same controller. During the creation of a new robot class, IP address and the port for the arms are needed. This starts a TCP/IP socket connection with the robot. This protocol enables communication trough the two devices. TCP controls the flow of data to prevent loss of data and makes sure the data arrives in sequence. The rest of the functions are divided into Set, Get and Various functions.

6.3 Robot library functions

These functions provide the instructions to control different aspects of the robot. It sends strings which contain two parts. The first variable contains a value which identifies the instruction. The second variable contains the values to execute the instruction. These variables are separated by a "/" for which the robot identifies the different parts.

For any "Get" function, the first variable is; query. This indicates that the computer is "asking" for some information. The second variable indicates which information is being requested. For instance "gcart" returns the Cartesian Coordinates.

So a possible Get command is "query/[gcart];".

The Set functions have more variables in the first part. For instance, possible first variables are; "gope" for gripper open, "orient" for the gripper orientation and "offset" for offset movement.

An example of a Set command is "offset/[30,-20,10,5,15,25];. This command moves the arm from its current position to a position relative to the current position. It moves the arm in X, Y, Z direction in 30mm, -20mm and 10mm. The rotation is also relative to the current position in Rotational X, Rotational Y, Rotational Z in 5, 15 and 25 in Euler angles.

6.3.1 Various functions

The various functions are the neither-get and set functions. These functions provide the basis for the other get and set functions to work. It provides the sending and receiving of commands from the computer and ensures a stable connection during periods of long inactivity. This bypasses the problem in which the robot loses the connection based on a time-out.

6.3.2 Manufacturer independent programming

The software is designed in such a way, that all of the programs can run separately from each other. The c# code sends the messages to the robot, which then decode into the specific ABB language. The last part is manufacturers dependent, which only works on ABB robots. This implies that it can run on the stock operating system provided by the manufacturer. But if the controller needs to control a robot from a different manufacturer, like KUKA, only the code on the robot needs to be programmed. This means that the same machine learning algorithm works for both the ABB and KUKA robot. A graphical explanation is given in appendix 11. The division by the different programming languages is marked with the different colours.

6.4 Performance

In order to validate whether the new controller is capable of replacing the current controller, it needs to perform in certain areas just as well as the current controller. These aspects measure the performance of the robot. Several aspects need to be measured in order to give a clear description of the performance of the controller and robot.

- Accuracy

  Whether the robot is able to perform movements with at least the same accuracy.

- Repeatability

  Maintaining the accuracy over several movements.

- Speed

  Determining if the robot is able to perform the same or more tasks in the same timespan.

- Reliability

  The likeliness that a certain action is completed successfully.

- Availability

  The period of which the robot is able to operate correctly.

- Maintainability

  The ability to provide maintenance to the robot and controller.

- Integrity

  The prevention of improper system modification, like bad software updates or wrongfully retrained the controller.

These aspects help to indicate the performance of a controller and test certain aspects of the whole system.

6.5 Future development

The current prototype shows that it is possible to control an IR in real time with a regular computer. It shows that it is possible to connect extra sensors to the computer to add functionality to the controller to add different control methods. For future development, the machine learning algorithm needs to be implemented. The yellow blocks in appendix 10 shows the parts which still needs to be developed. The Accord library could function as a machine learning library for this controller. There are already several algorithms which can be used for object recognition, which are also written in c# and can therefore be implemented quickly. The Tensorflow library is a widely used library with more examples including state of the art CNN networks. Even though this library is not available in c#, it can be implemented in this controller by a parser, which sends data over two different languages. This language has a larger community than the Accord library. Advanced machine learning networks are more likely to be published in Tensorflow, which means the newest algorithms can be used in this controller.

7 Conclusion

For humans to cooperate with robots, lots have to be changed on how robots are applied in the current situation. While the market is shifting towards cooperation between humans and robotics, the approach of robotic control did not change. So for cooperation to be a success, robots have to think for themselves and anticipate on the environment and user. This can be achieved by machine learning algorithms. These are currently improving, improving the possibility of using these algorithms for human-robotic cooperation. However, it requires more input from the environment and user and a different approach of learning the robot the desired tasks. Specific algorithms for these tasks have to be developed to handle all the different kinds of inputs from the environment to make decisions while maintaining the advantages of industrial robots, such as reliability and quality of the process.

8 Discussion

The whole approach of better cooperation between humans and robots is to give robots more human-like attributes. This can be seen in the design of several cooperative robots, giving it human eyes, like the Rethink Robotics Sawyer, or a human posture, like the ABB YuMi. A comparable development has been made during the design of aircraft. Designers first tried to mimic the movements of birds to create lift. Aircraft nowadays use different techniques in order to produce the same result. This indicates that mimicking human or natural behaviour might not be the solution for human-robotic cooperation. It may require a different method. A possible way could be of excluding the human, and let machine learning algorithm learn tasks by trial and error, known as reinforcement learning. This might take over complex human tasks, eliminating the need for human-robotic cooperation.

References

[1] Berkeley 3-d object dataset. http://kinectdata.com/.

[2] Bias-variance tradeoff in machine learning — learn opencv. http://www.learnopencv.com/bias-variance-tradeoff-in-machine-learning/.

[3] Coco - common objects in context. http://mscoco.org/.

[4] Kinect v2 depth camera calibration — three constants. https://threeconstants.wordpress.com/2014/11/09/kinect-v2-depth-camera-calibration/.

[5] Openai gym. https://gym.openai.com/docs.

[6] Understanding convolutional neural networks for nlp – wildml. http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/.

[7] Basic concepts in machine learning. http://machinelearningmastery.com/basic-concepts-in-machine-learning, Sep 2016.

[8] Tensorflow. https://www.tensorflow.org/, 2017.

[9] ABB. robotics open abb. https://github.com/robotics/open_abb, Dec 2016.

[10] Ethem Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning series)*. The MIT Press, 2009.

[11] Chandrayee Basu, Qian Yang, David Hungerman, Mukesh Singhal, and Anca D. Dragan. Do you want your autonomous car to drive like you? In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction - HRI*. ACM Press, 2017.

[12] Anca D. Dragan, Shira Bauman, Jodi Forlizzi, and Siddhartha S. Srinivasa. Effects of robot motion on human-robot collaboration. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction - HRI*. ACM Press, 2015.

[13] Yan Duan, Marcin Andrychowicz, Bradly C. Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. *CoRR*, abs/1703.07326, 2017.

[14] Madeline Gannon. Robo op. http://www.madlab.cc/robo-op, 2016.

[15] IBM. Machine learning algorithm != learning machine. https://www.ibm.com/developerworks/community/blogs/jfp/entry/Machine_Learning_Learning_Machine?lang=en, Apr 2016.

[16] Fraunhofer IPA. Eu project smerobotics: Versatile robots for the digitized industry, 2016.

[17] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, L. Cehovin, G. Fernandez, T. Vojir, G. Hager, G. Nebehay, R. Pflugfelder, A. Gupta, A. Bibi, A. Lukezic, A. Garcia-Martin, A. Saffari, A. Petrosino, and A. Solis Montero. The visual object tracking vot2015 challenge results. In *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, pages 564–586, Dec 2015.

[18] Eric Meisner, Volkan Isler, and Jeff Trinkle. Controller design for human-robot interaction. *Autonomous Robots*, 24(2):123–134, 2008.

[19] Pramila Rani, Nilanjan Sarkar, Craig A. Smith, and Leslie D. Kirby. Anxiety detecting robotic system towards implicit human-robot collaboration. *Robotica*, 22(1):85–95, jan 2004.

[20] Kyle Strabala, Min Kyung Lee, Anca Dragan, Jodi Forlizzi, Siddhartha Srinivasa, Maya Cakmak, and Vincenzo Micelli. Towards seamless human-robot handovers. January 2013.

[21] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[22] Matthew P. Wallen, Sjaan R. Gomersall, Shelley E. Keating, Ulrik Wisløff, and Jeff S. Coombes. Accuracy of heart rate watches: Implications for weight management. *PLOS ONE*, 11(5):e0154420, may 2016.

[23] WIRED. 4 things robots need to learn before working with humans. https://www.wired.com/2017/04/4-things-robots-need-learn-working-humans/, Apr 2017.

9 Appendix

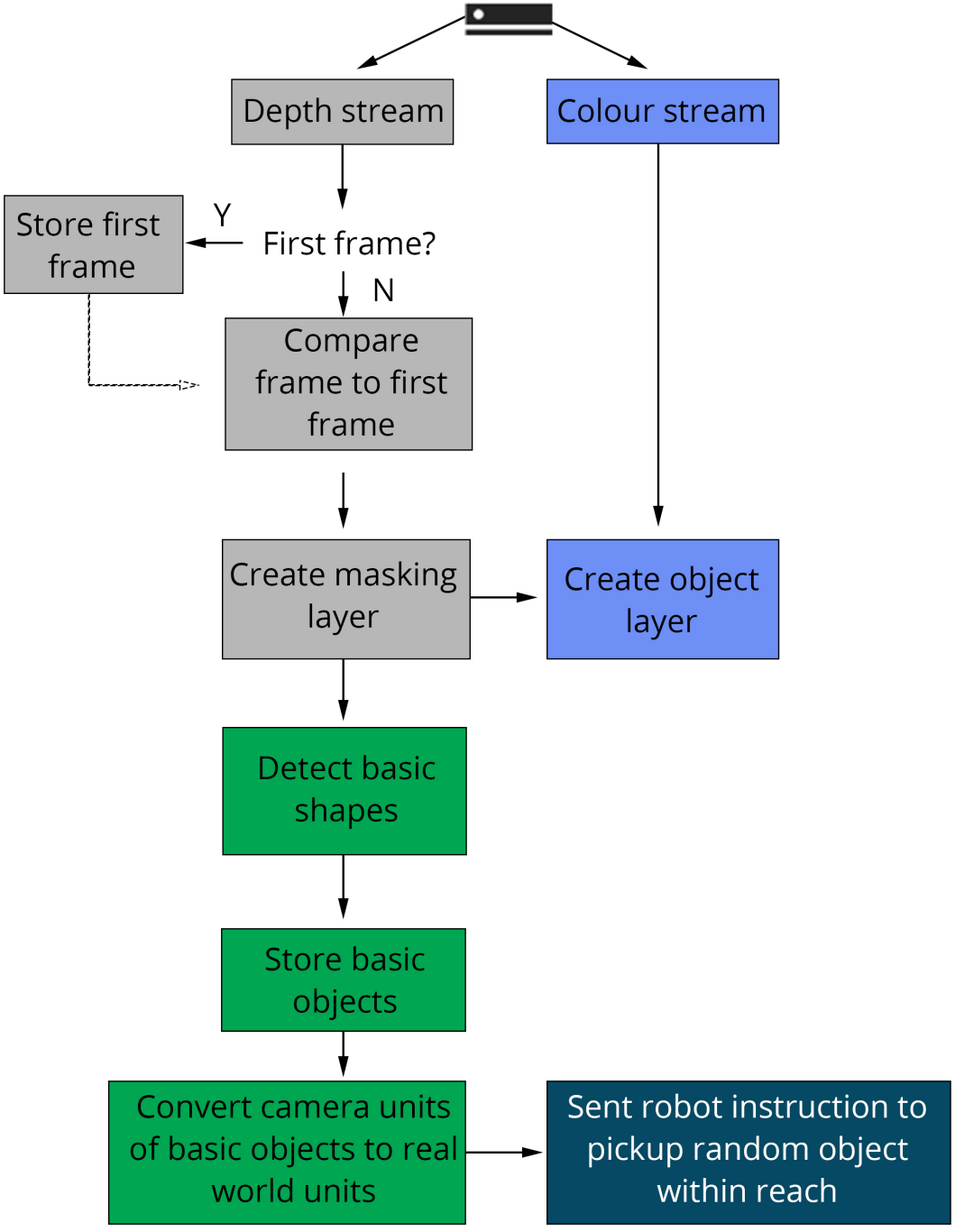

Figure 9: Block diagram of the current controller. Gray shows the depth frame processing. Light blue the colour frame processing. Green shows the OpenCV part and the blue-gray part the c# robot controller.

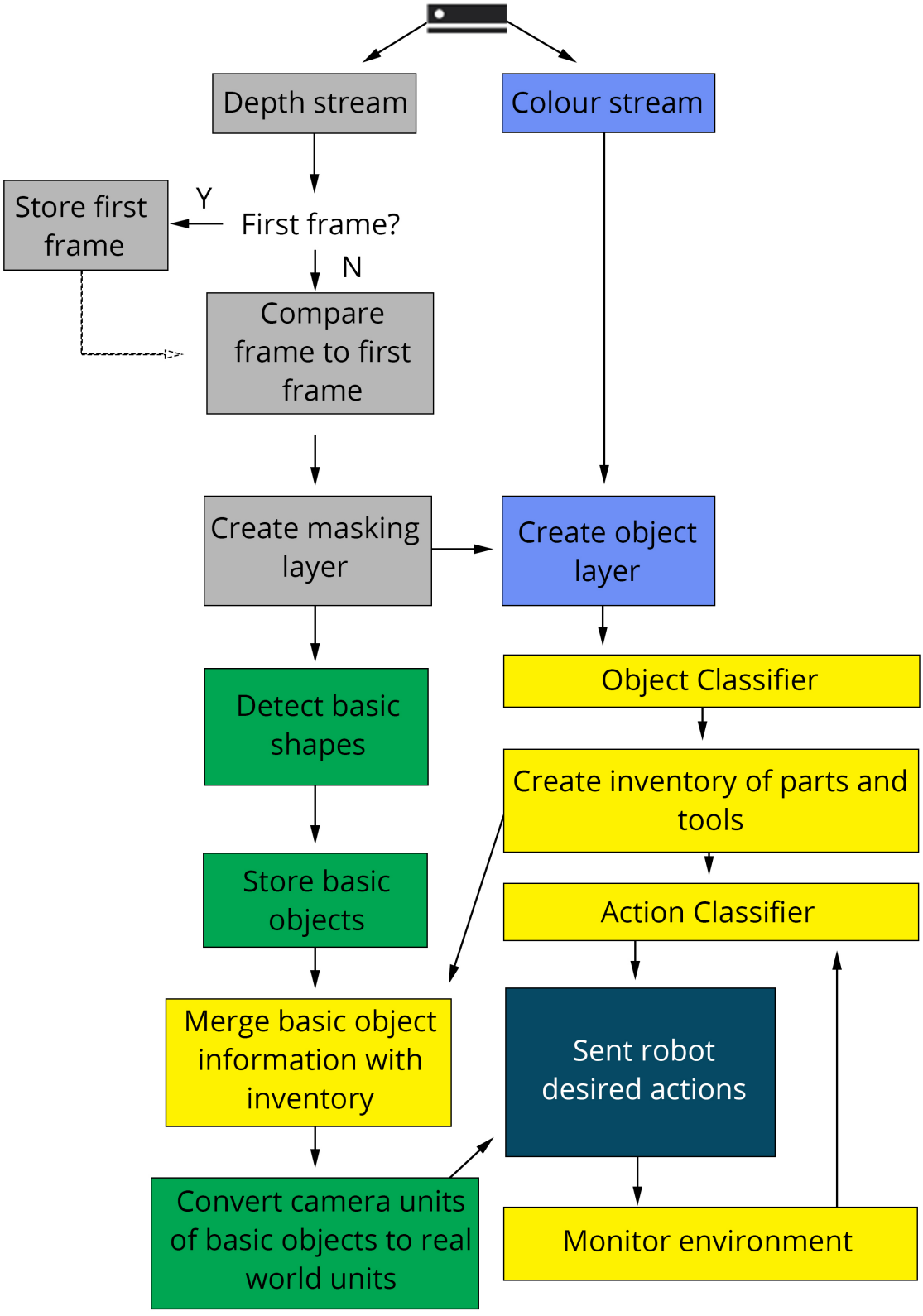

Figure 10: Block diagram of the desired controller including machine learning, shown in yellow. This part is not yet implemented in the controller.

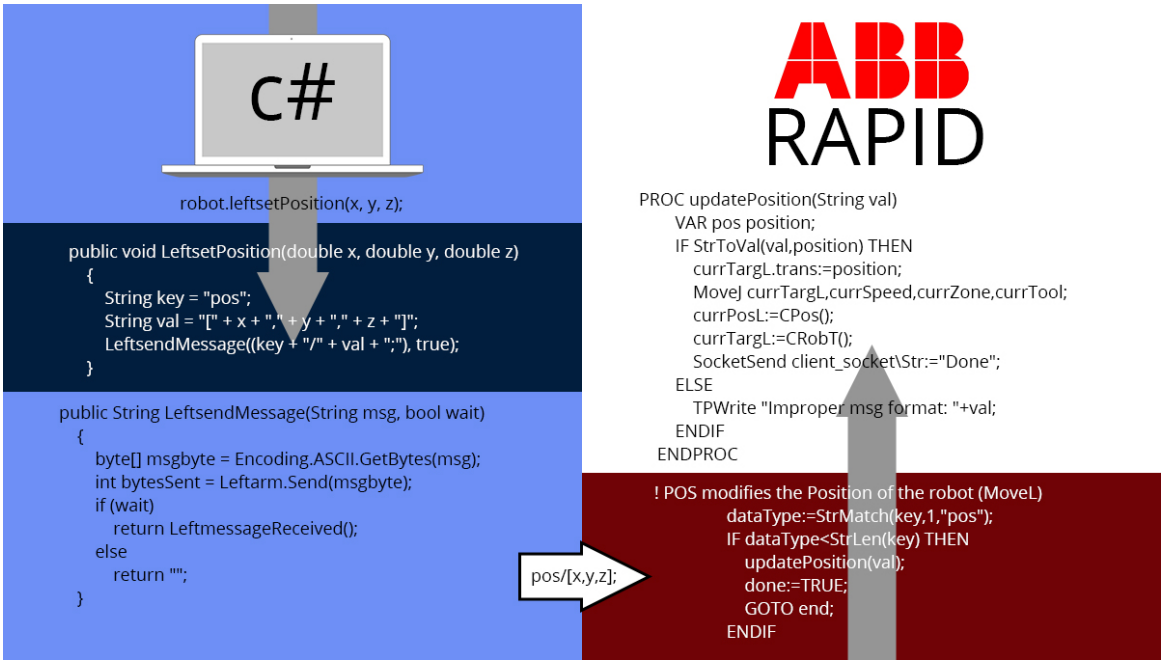

Figure 11: Graphical software explanation. The software is divided into two parts, the C# and RAPID part. The message shared is in the white arrow

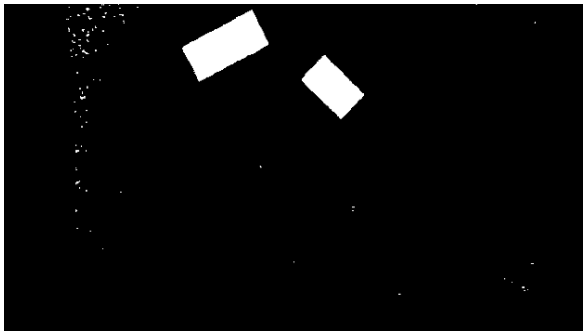

Figure 12: Masking Layer

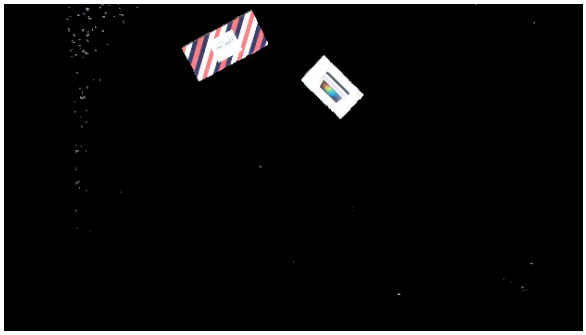

Figure 13: Object Layer

Figure 14: Canny Edge Detection

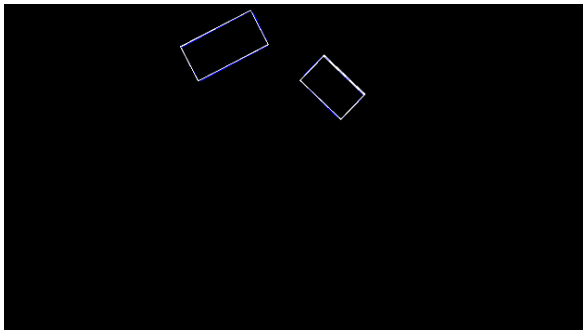

Figure 15: Basic Objects

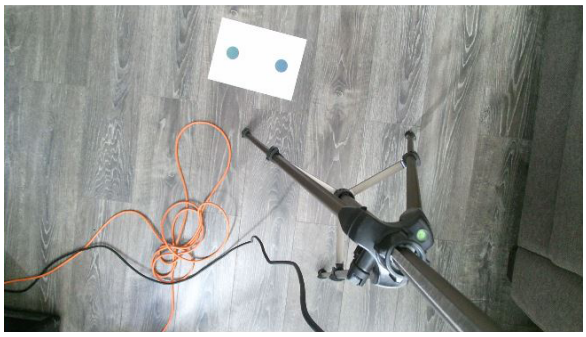

Figure 16: RGB Input

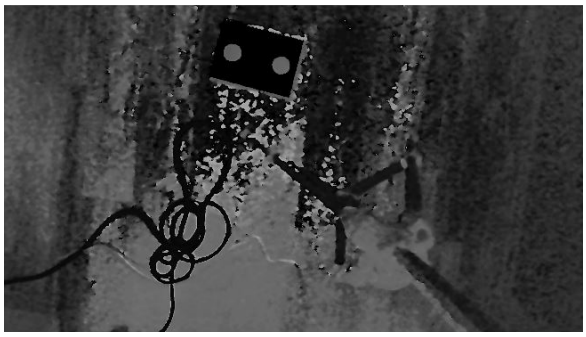

Figure 17: Hue layer

Figure 18: Saturation Layer

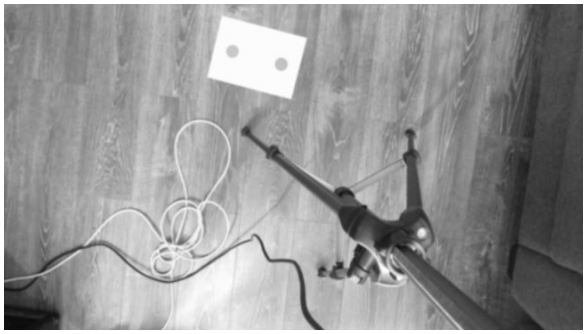

Figure 19: Value Layer

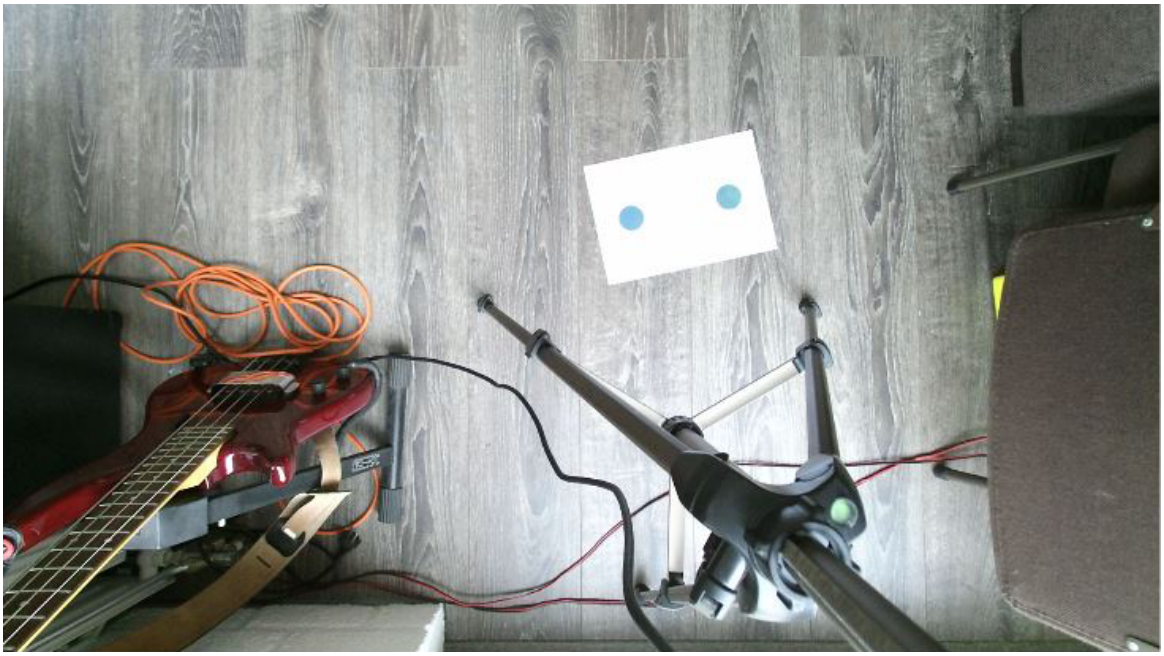

Figure 20: Input Image

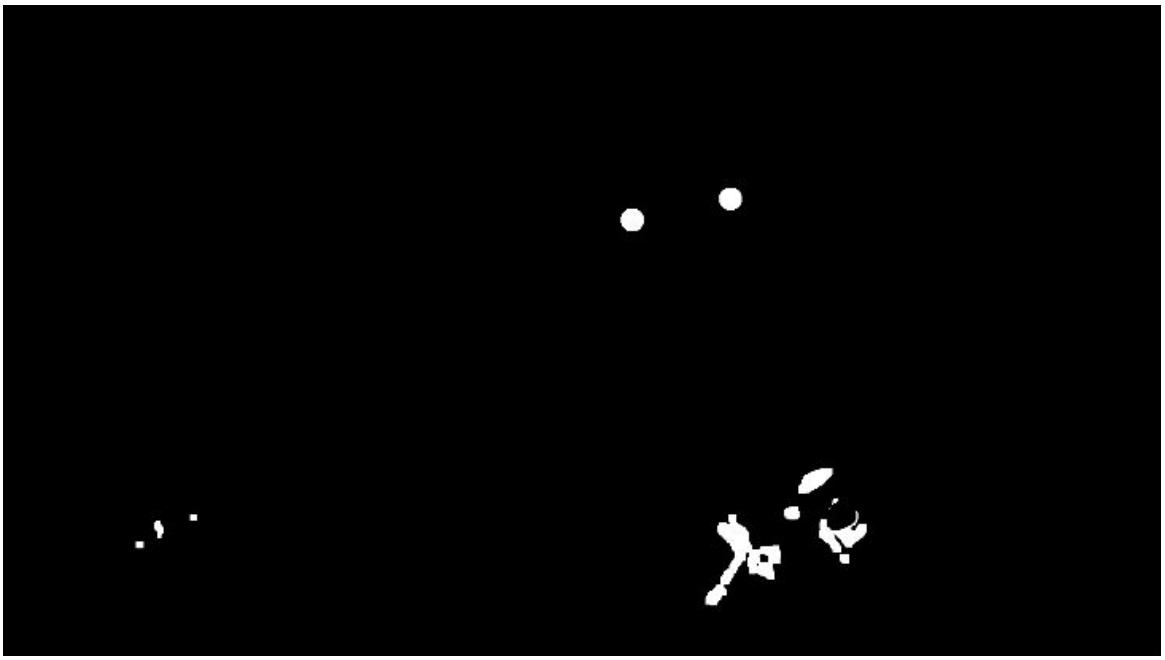

Figure 21: Hue and Saturation Mask

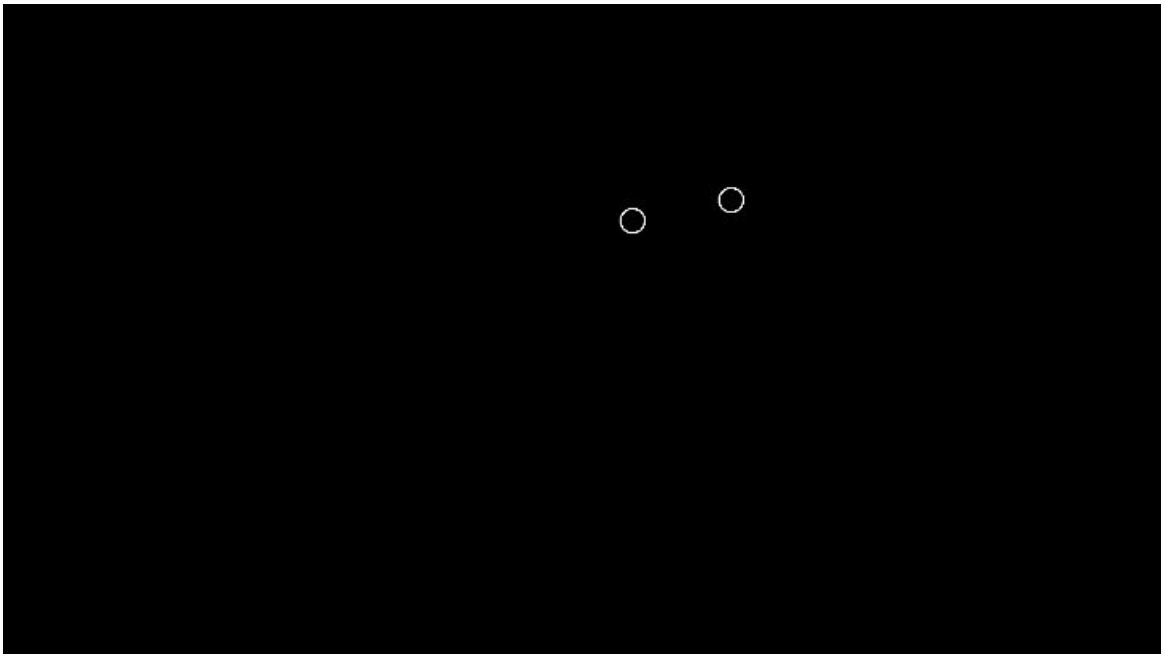

Figure 22: Circle Detection
red

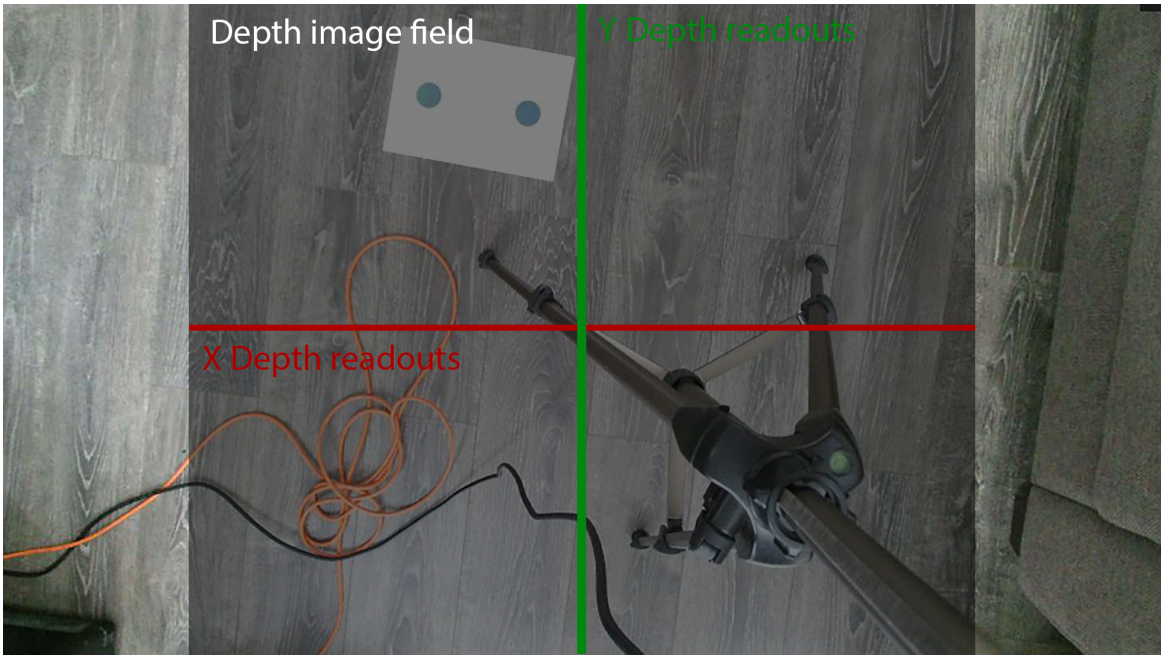

Figure 23: The positions of the depth readouts used in the graphs below.

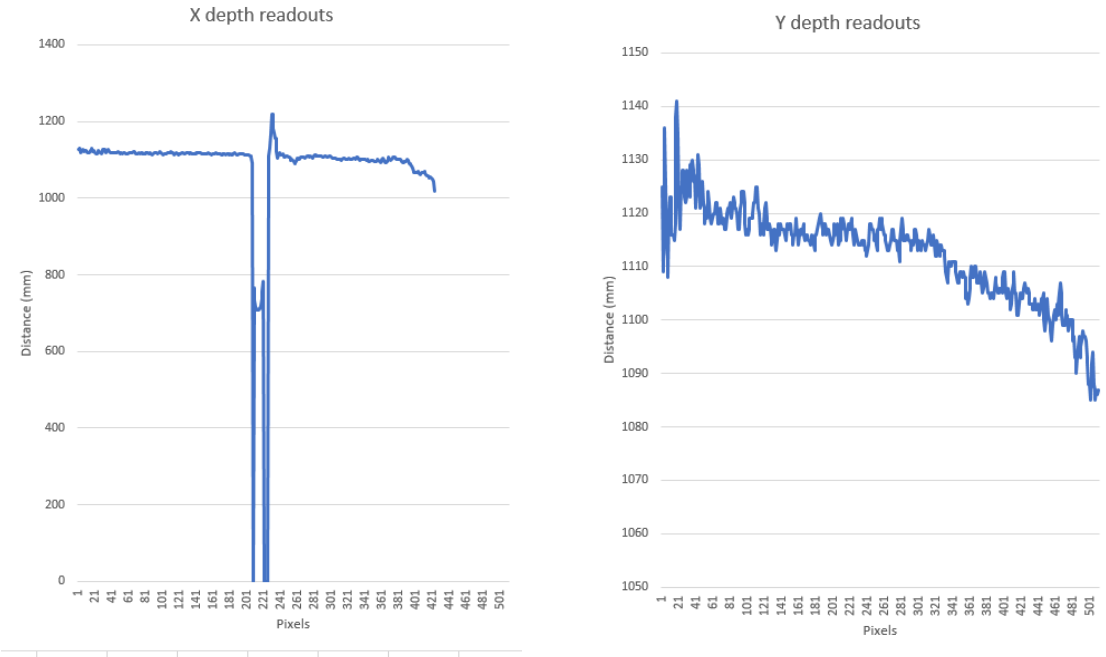

Figure 24: Depth readouts. The values are in millimetres.

Figure 25: Prototype test setup

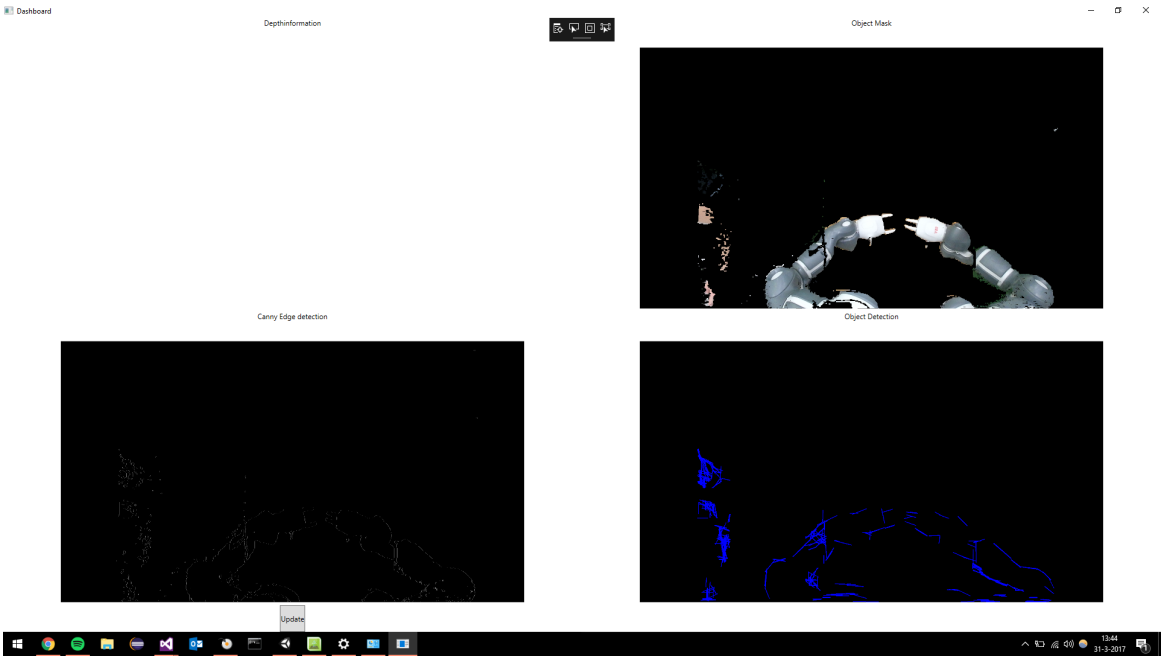

Figure 26: Picture of the images on the test setup

```
public static Quaternions rotationConversion(double yaw, double pitch, double roll)
{
    yaw *= Math.PI * yaw / 180;
    pitch *= Math.PI * pitch / 180;
    roll *= Math.PI * roll / 180;

    double rollOver2 = roll * 0.5f;
    double sinRollOver2 = Math.Sin((double)rollOver2);
    double cosRollOver2 = Math.Cos((double)rollOver2);

    double pitchOver2 = pitch * 0.5f;
    double sinPitchOver2 = Math.Sin((double)pitchOver2);
    double cosPitchOver2 = Math.Cos((double)pitchOver2);

    double yawOver2 = yaw * 0.5f;
    double sinYawOver2 = Math.Sin((double)yawOver2);
    double cosYawOver2 = Math.Cos((double)yawOver2);

    Quaternions result = new Quaternions();

    result.W = cosYawOver2 * cosPitchOver2 * cosRollOver2 +
                                        sinYawOver2 * sinPitchOver2 * sinRollOver2;
    result.X = cosYawOver2 * sinPitchOver2 * cosRollOver2 +
                                        sinYawOver2 * cosPitchOver2 * sinRollOver2;
    result.Y = sinYawOver2 * cosPitchOver2 * cosRollOver2 -
                                        cosYawOver2 * sinPitchOver2 * sinRollOver2;
    result.Z = cosYawOver2 * cosPitchOver2 * sinRollOver2 -
                                        sinYawOver2 * sinPitchOver2 * cosRollOver2;

    return result;
}
```

Figure 27: Euler angles to Quaternion calculation