ROBOTICS AND MECHATRONICS

AUTONOMOUS NAVIGATION FOR THE PIPELINE INSPECTION ROBOT "PIRATE"

T.V. (Tim) Kesteloo

MSC ASSIGNMENT

November, 2020

053RaM2020
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

Summary

Smart pipe inspection robots are used more and more to perform quality control from inside pipelines. The "Pipe Inspection Robot for AuTonomous Exploration" PIRATE, developed at the University of Twente, is a snake-like pipe inspection robot that can traverse such pipe networks by clamping itself between the pipe walls of varying pipe diameters and pushing itself along with its wheels. A rotational module allows the PIRATE to align with bends and junctions encountered while traversing the pipe network.

This research increases the autonomy of the PIRATE by means of the newly developed control layer that uses the existing PIRATE motion primitives. The control layer is a centralized approach to Path Planning and Motion Planning, responsible for generating a path in a known three dimensional pipe network, and converting the path to the appropriate motion sequences for the PIRATE.

Path Planning uses a graph-node based map for the pipe network where each node is an elbow bend or T-junction in the network. These landmarks are connected to one another with straight pipes. A shortest path from any starting landmark to any goal landmark is calculated using the A*-algorithm, using the 3D Euclidean distance between the traversable nodes as cost-function, and the 3D Euclidean distance between the node and the goal as heuristic function. The path consists of motions for a wheeled vehicle with regards to the frame of the pipe network.

Motion Planning uses the state of the PIRATE robot to evaluate which move is required next. This converts the motions provided by Path Planning from a wheeled robot perspective to motions that are applicable to the PIRATE. The combination of Path Planning and Motion Planning allow for calculating the quickest path with the A*-algorithm. Using a modified cost-function that penalizes rotating the PIRATE, a path to the goal landmark requiring the least amount of rotations for the PIRATE is calculated.

Tests in the simulation environment V-REP show that a digital model of the PIRATE can be instructed to move through a known (mapped) pipe network that consists of challenging sections like vertical pipes. The test uses a simulated corner detection method which allows the PIRATE to detect the subsequent landmarks and their orientation. The PIRATE receives a sequence of motions to perform once a landmark has been detected in order to continue the mission. In case the robot becomes stuck, autonomy can be temporarily deactivated and allow the operator to manually instruct the PIRATE to perform motion primitives.

Further research should focus on combining the work on Corner Detection, Feature Extraction, and Mission Control for the physical PIRATE robot, fitting the physical model with the appropriate sensors to accommodate them. Additionally, more fine-grain control is required for the rotational module of the PIRATE to (re-)align for corners.

Contents

1 Introduction

1.1 Context

Current methods of quality monitoring for petrochemical pipe networks include leak searching (or 'sniffing') above ground, and robots inside the pipes below ground. Such smart Pipe Inspection Gages (PIG) are generally large cylindrical robots that propel themselves through the pipe network to perform inspections. However, research from 2014 showed that around 40-50% of all oil and gas pipelines are "unpiggable" (Mazraeh et al., 2014) due to (among other things) differing sizes of pipes in a network, sharp corners, or insufficient friction for a PIG to travel in the pipe.

UTwente's Robotics and Mechatronics group (RaM) has worked on the "Pipe Inspection Robot for AuTonomous Exploration" (for short PIRATE) since the beginning of 2006 (Dertien, 2006). The PIRATE is a snake-like robot designed for industrial pipe inspections by clamping itself between pipe walls of different pipe diameters. The robot consists of multiple modules connected with rotational joints that enable the robot to bend and deform, a rotational section in the middle of the design which allows the robot to turn the front and rear section, and wheels in each joint which allow the robot to drive itself forward and backwards. A sideview of the robot is depicted in Figure 1.1. This design allows for higher mobility to overcome challenges such as differing sizes of pipes and sharp corners, making it possible to traverse the "unpiggable" pipe networks.

1.1.1 Recent work

Several years have been spent on creating the means for the PIRATE robot to become autonomous. Garza Morales (2016) did so by defining smaller tasks, such as clamping to a pipe wall, and taking several types of corners, called "Partially Autonomous Behaviour" (PABs). Following this, Hoekstra (2018) focused on creating a software architecture based on the RobMoSys framework, which led to the introduction of "*Sequences*", which are consecutive combinations of PABs. Geerlings (2018) designed a high-level control layer that would reduce the amount of slipping and chance of falling when the robot would drive through the pipe, particularly in vertical sections. Kleiboer (2019) researched what is needed to identify the complex pipe structures using an on-board sensing system. This has led to the recommendation to integrate a LiDAR as a means for feature extraction inside the pipe.The latest research done for the PIRATE is the work done by Tersteeg (2020), who designed a preemptable task architecture, allowing for stopping or re-executing tasks whenever they fail or took too long to complete. Par-

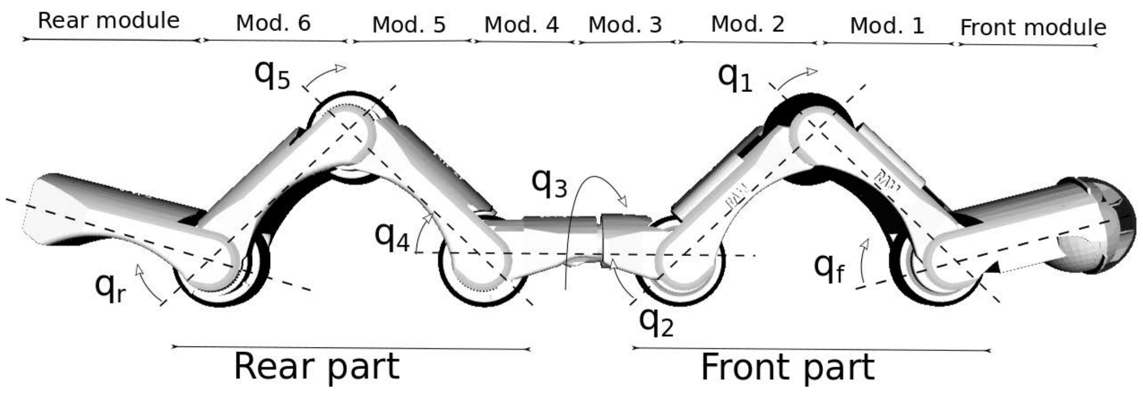


Figure 1.1: Sideview of the PIRATE model. The robot model is divided into two parts, separated by the rotational module in the middle (q_3). Each rotational joint ($q_f, q_1, q_2, q_4, q_5, q_r$) sits on top of a wheel.

allel to this research, Vijaykumar (n.d.) is working on Simultaneous Localization and Mapping (SLAM) for the PIRATE robot.

1.2 Problem Description

The PAB Sequences allow the PIRATE to perform several more complex motions to traverse pipe networks. In order to make the system more autonomous, an extra layer of control has to be constructed which uses the available information to determine which actions to take next. Actions could be described as behaviour when encountering an object or corner, i.e. a sequence of motions required to pass by the object. This should conclude in the ability to perform missions, ranging from exploring a pipe network while determining whether pipe quality suffices, to going to a specified location on the map to perform an inspection there.

To clarify what the scope of this research will be, we formulate what we would like the PIRATE to be able to achieve in the future. The vision is *to allow the PIRATE robot to (a) navigate autonomously (b) through both known and unknown configurations (maps), (c) taking complex corners (d) through different pipe sizes (e) safely, while (f) avoiding collisions with other robots or objects in the pipe, which enables the PIRATE robot to (g) perform inspections in said pipe network.*

1.3 Goal

This research will focus mainly on the subject of *(a) autonomous navigation* and partly on *(b) navigating through known and unknown pipe networks*, as close cooperation will be required with the SLAM research by Vijaykumar (n.d.) to traverse unknown networks. Goals *(c)* and *(d)* have already been tackled by previous research. Goal *(e)* has been present since the beginning and all existing software layers take this into account in some form. Goals *(f)* and *(g)* are left for future implementation, although the aim is to design the new control layer with these in mind.

To achieve these two goals, we formulate the following **Design Objective**: *To construct an extra control layer for the PIRATE that deals with navigation and planning to allow for autonomous control.*

A theoretical framework needs to be constructed in order to develop a new control layer. The following **questions** are answered before starting development.

- Which motions can the PIRATE perform and how do these translate to the motions of a typical wheeled robot vehicle?
- Which input information (sensory and provided data) is minimally required for the control layer to enable a PIRATE to drive autonomously in a pipe network?
- How can 3D navigation for the physical PIRATE robot be achieved in a pipe network?

1.4 Approach

In order to answer all questions and achieve the objective, several topics are researched before a control layer is created.

First the components and systems that make up the PIRATE and its software are researched. It is identified what the PIRATE robot is already capable of and how this translates into motions in a 3D pipe network. These are put into context with available robots similar to the PIRATE to better understand what the challenges for these robots are.

Afterwards research is done to gain a better understanding of possible pipe networks; What elements and bends do they consist of, and what other problems can autonomous robots in pipe networks come across? It is important to understand how these components are used in practice, to have an understanding of what the minimal requirements for the system must become while making sure that future development can complete the system without interference of previous functions.

Consequently, existing methods of driving and navigating robots autonomously through both known and unknown systems are investigated. The research on Automated Guided Vehicles (AGVs) and fully autonomous Self-Driving Vehicles (SDVs) are explored to better understand how their perception of the environment contributes to the decision making process.

Once these have been researched, a control framework is chosen based on best-practices and existing solutions from literature, which is implemented.

1.5 COVID-19 and testing implications

Due to the global pandemic and national health guidelines at the time of this project, the physical PIRATE robot cannot be accessed or used for testing. To overcome this, testing the implementation of the autonomous control layer is done in the simulation software V-REP (now CoppeliaSim), which is further explained in Section 2.5. Nonetheless, design decisions and development are conducted with the physical robot in mind.

1.6 Outline

Chapter 2 explores the questions and the related research fields of the PIRATE, motion, navigation, and path planning. Based on these findings Chapter 3 analyses how these topics relate to the scope of this research, and chooses an approach for development of the control layer. Chapter 4 describes the development of the new components of the control layer and simulation environment. The resulting control layer and final experiments are shown in Chapter 5, after which the chosen solutions and newly found challenges are discussed in Chapter 6. Finally, Chapter 7 answers the related questions and concludes the proposed design objective, resulting in recommendations and considerations for future research.

2 Background

2.1 PIRATE System

The PIRATE robot (Figure 1.1) and its systems are described in this section to give an overview of the robot and its capabilities. First the design of the robot with regards to its motions is explained. Lastly, the software architecture is explained, and the flow of information and systems required to distribute it is given.

2.1.1 PIRATE Motions

The PIRATE robot consists of multiple modules connected with rotational joints, a rotational section in the middle of the design which allows the robot to turn the front and rear sections, and wheels in each joint which allow the robot to drive itself forward and backwards, as depicted in Figure 1.1. The PIRATE can turn around its own axis with the rotational module, given that one part is clamped inside a pipe while the other is relaxed. The previously defined PAB Sequences allow for turning this module either 90 degrees left or right, and performing such an action twice allows it to turn 180 degrees. The existing PAB Sequences include but are not limited to, EnterPipe, AlignLeft, AlignRight, TurnForward, TurnBackward, and three TakeTurn commands.

Alignment of the robot before a TakeTurn Sequence is important, as this is designed with the proper alignment of the robot in mind. Due to the turning required to be aligned before taking a turn means that the robot is not always belly-down, and Geerlings (2018) even proposes the PIRATE behaviour to prefer not being belly-down when traversing the network as to avoid driving through any residue that would be at the bottom of the pipes. This constant turning and twisting of the PIRATE means that its frame of reference changes at each corner taken. This also means that any pre-planned motion for the PIRATE should take this change of frame into account when calculating the PIRATE actions to traverse a pipe network.

Additionally, extra sensors could be used to verify the frame and rotation of the PIRATE. It is useful to calculate after each move whether the PIRATE is properly aligned before resuming a Sequence. The absence of these sensors can result in slight misalignment at each performed motion, possibly leading to wrong decisions or even over time critical failures. However, since the Sequence commands do not currently allow for rotating on such a fine-grain level (only 90 degree rotations), realignment after such an observation is currently not possible.

2.1.2 Software Structure

The existing PIRATE code has been iteratively created by multiple contributors. Based on the layered software architecture by Broenink et al. (2010) shown in Figure 2.1, several modules/nodes have been created that handle each of the tasks. A visual representation of this implementation is shown in Figure 2.2 by Tersteeg (2020). We shortly explain the implementation of the upper two sections, Supervisory control and Sequence control, as to better establish the starting situation for this project.

The Sequence Control sections consists of two classes, namely Movement and PAB (Partially Autonomous Behaviour). A PAB is a series of movements that each limb has to take in order to do a 'basic' task, like clamping to a pipe, or taking a corner in a certain pipe diameter. This is done by sending a sequence of messages to the Movement node (Tersteeg, 2020). The Movement node is responsible for sending these commands to the hardware through its MOV messages, which defines which motor is actuated to what extent. The Supervisory section is connected to the Sequence layer through the Sequence class. Certain combinations of PABs have been categorised into a set of actions that can be taken, which are the actions from the

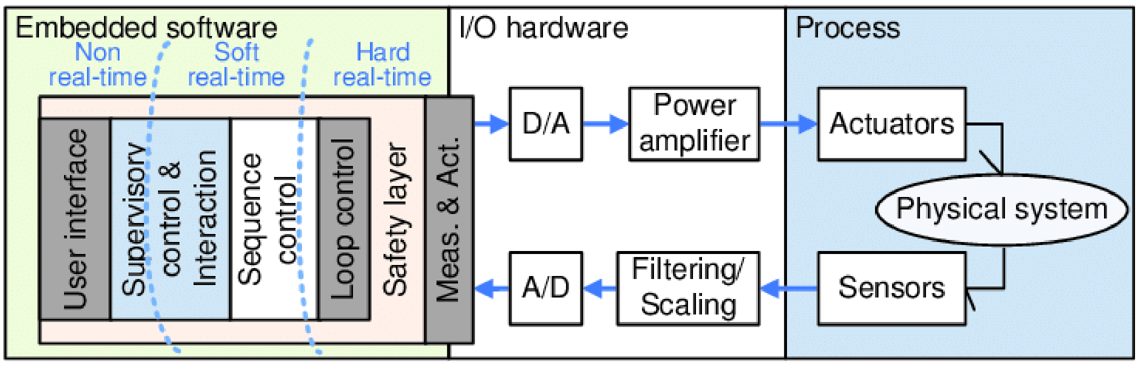


Figure 2.1: Embedded Control System and its Software Architecture (Broenink et al., 2010)

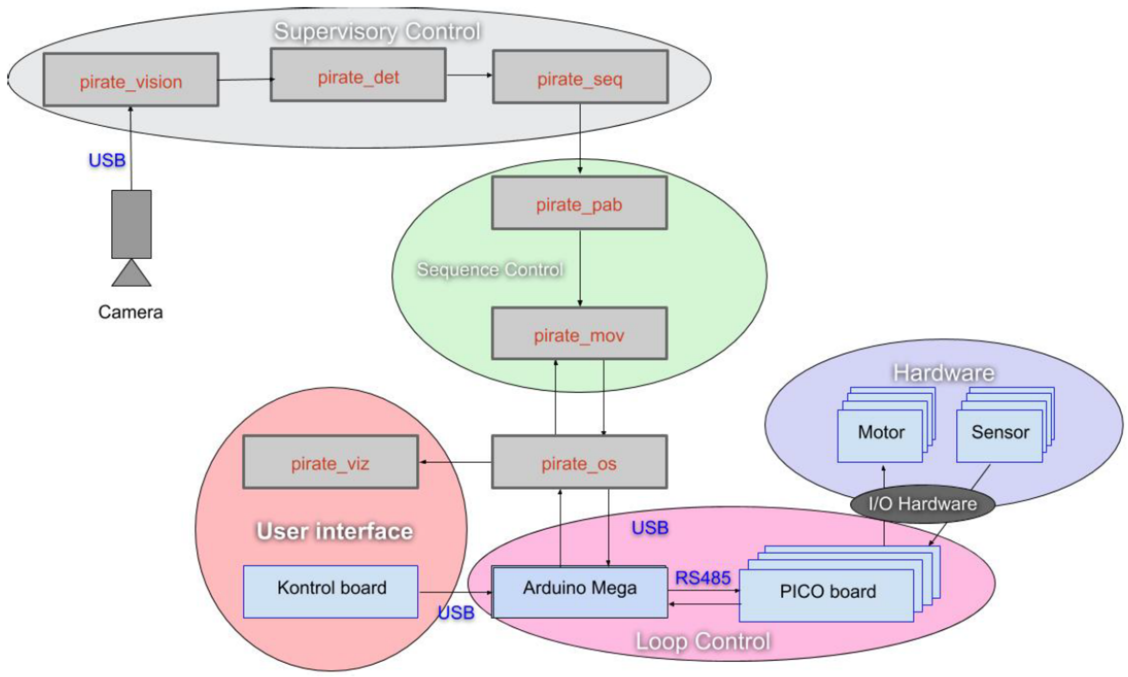


Figure 2.2: PIRATE Software Architecture (Tersteeg, 2020)

Sequence class. These actions currently encompass *EnterPipe, TurnForward, TurnBackward, AlignLeft, AlignRight, TakeTurnEarly, TakeTurnMid, and TakeTurnLate.* This set of movements allows the robot to drive through a pipe network by clamping to a pipe, driving through it, and take corners. The two subsequent classes are Detection and Vision. The latter is connected to the camera input and converts these into coordinates using the OpenCV library. These coordinates are interpreted by the Detection class, which decides based on the coordinates whether a corner is close by, is halfway through the bend, or has reached the end of the corner, and sends messages to the Sequence node accordingly.

Robot Operating System

The existing software architecture is developed using the Robot Operating System, or ROS . ROS is *a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms*[1]. This open-source platform releases a new version of core packages almost on a yearly basis, is compatible with both C++ and Python, and requires a Linux distribution.

[1]https://www.ros.org/about-ros/

The ROS architecture uses the publish-subscribe messaging pattern. This means that each class (or 'node') can both send and receive data ('topics') to and from other nodes without requiring specific knowledge of who the senders or receivers are. These messages are recorded into so called ROS-bags, which can be used to replay past scenarios for the use of reproduction and debugging.

Previous research opted for utilising this platform, and based on its extensive knowledge-base and existing software solutions it seems suitable to this research. Because of this, the newly developed control layer is also written using ROS.

2.2 Similar pipe inspection robots

The summary by Mills et al. (2017) makes a distinction between 8 basic designs for pipe inspection robots, of which combinations are often made. Of these categories, the PIRATE belongs to the snake variant with a 2-plane wall pressing mechanism. In this category the writers include the PipeTron series by HiBot, Tokyo (Debenest et al., 2014), the Explorer series from Carnegie Mellon University, America (Schempf and Vradis, 2003), a design by Kanagawa University, South Korea, and another design by the Czech Technical University. The writers acknowledge that the snake wall-pressing design is an advantageous design due to its size and adaptability, like being able to overcome pipe-size changes and even surpassing valves (pipe segment B and C in Figure 2.3).

More recent examples of snake-like pipe inspection robots that are in development are the AIRo series (Kakogawa and Ma, 2020), all the way up to version 2.5 which would have been presented at SXSW 2020, Austin, Texas, USA [2], and new designs like the Omnidirectional Tractable Three Module Robot (Suryavanshi et al., 2019), and the Cylindrical Elastic Crawler Robot (ya Nagase et al., 2018).

Each of these designs are aimed at achieving the highest mobility without compromising usability or size. However, none seem to have found the best design yet, as new designs are still being researched, trying to overcome the next obstacle. None of the non-swimming robots summarized by Mills et al. (2017) can traverse all in-pipe geometries, travel far distances, and handle all pipe diameters.

2.3 Pipe Network components

The PIRATE is designed to drive through pipe networks, focusing on petrochemical plants. These, and other pipe networks, can generally be defined by the components they consist of. Figure 2.3 shows the most commonly encountered pipe segments, as defined by Mills et al. (2017). Previous research for the PIRATE has dealt mostly with (a) horizontal, (d) vertical, and (e) elbow sections. The original design by Dertien (2006) also defines the parameters for the (c) diameter reduction/change, and (f) T-junction, but recent work rarely uses these elements for testing. In order to make decisions about movement and direction in a pipe network, it is minimally required to include the (a) horizontal, (d) vertical, (e) elbow bends, and (f) T-junction segments to the set of traversable obstacles for this research.

[2] https://youtu.be/rk7l05f1nE8

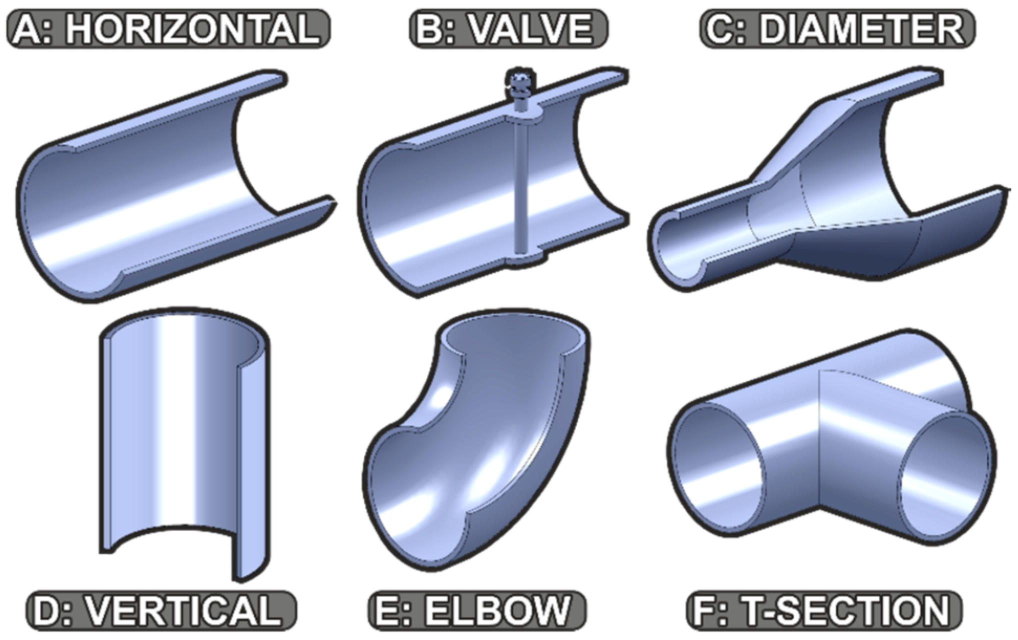


Figure 2.3: Most commonly encountered pipe bends and joints (Mills et al., 2017)

2.4 Decision making and Control

Autonomous vehicles on the road (SDVs) deal with incredible complexities, for example analyzing the trajectory of pedestrians, or whether a detected stop sign is applicable to the the situation of the vehicle[3]. An example of a framework that aims to deal with such complexities is Autoware[4]. The goal of this open-source platform is to enable self-driving mobility. A good example of the systems that are required to achieve this is shown in the publication by Kato et al. (2018). In Figure 2.4 they present their basic architecture for control and dataflow. The motion planning, in charge of creating feasible trajectories, is influenced by the decision module, which is responsible for estimating what other objects will do. This is done using several state machines, like the one shown in Figure 2.5.

For AGV systems, decision making often comes down to a scheduling problem; which robot is best used for task x at time y. Another aspect is the ability for a robot to change a mission on the fly based on the sensor readings of the vehicle, which can lead to a slight variation from the planned behaviour. Since it is hard to obtain such information from inside the pipeline networks, it becomes difficult to implement algorithms that prevent this from happening like Yan et al. (2017) with set warehouse distance parameters, or Li et al. (2010) using cyclic prediction algorithms based on traffic rules.

Ryck et al. (2020) looked at many AGV-systems and defined several challenges and systems that are present in each and should be dealt with accordingly. The five core tasks of an AGV system described are *Task Allocation, Localization, Path Planning, Motion Planning*, and *Vehicle Management* as shown in Figure 2.6. *Task Allocation* is responsible for assigning a robot from the swarm to perform an operation, which is not relevant for our research, as a swarm of PIRATEs does not yet exist. *Localization* is the ability of the robot to accurately place itself in its environment. This too is outside of the scope of this research and will be dealt with in the SLAM research for the PIRATE by Vijaykumar (n.d.), which is addressed in Section 2.4.2. *Path Planning* and *Motion Planning* are described respectively as static path planning and dynamic path planning. This means that the path planner is responsible for computing a collision free

[3]https://youtu.be/hx7BXih7zx8?t=513 ScaledML2020: AI for Full-Self Driving at Tesla
[4]https://www.autoware.org/

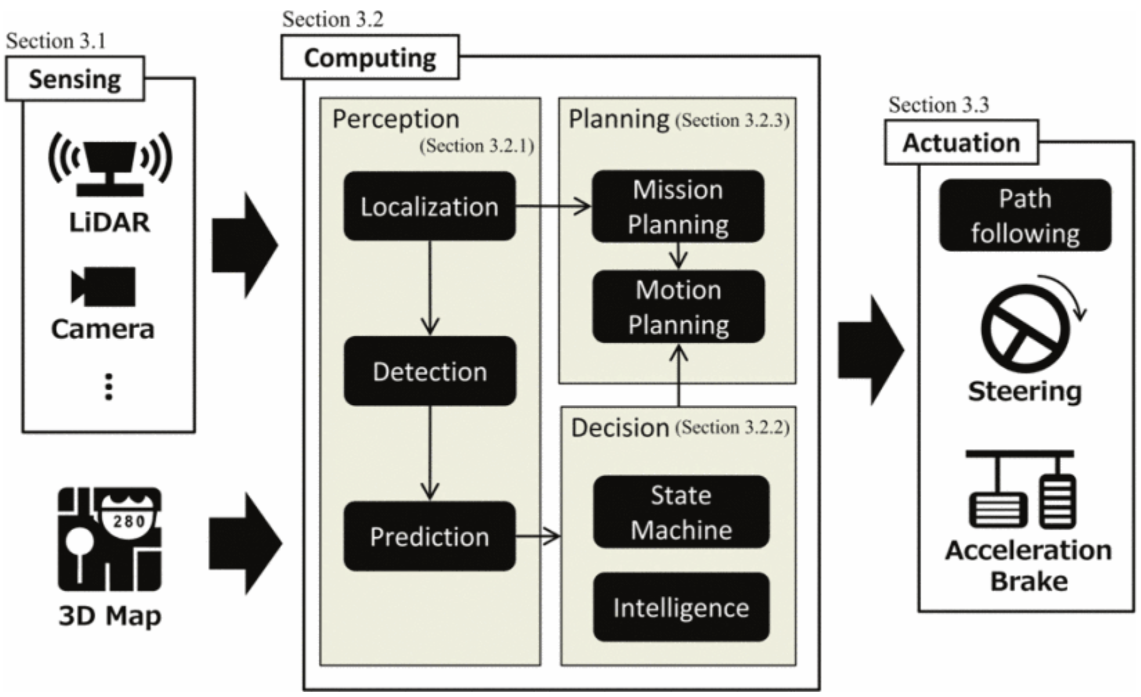


Figure 2.4: Basic information flow for autonomous vehicles (Kato et al., 2018)

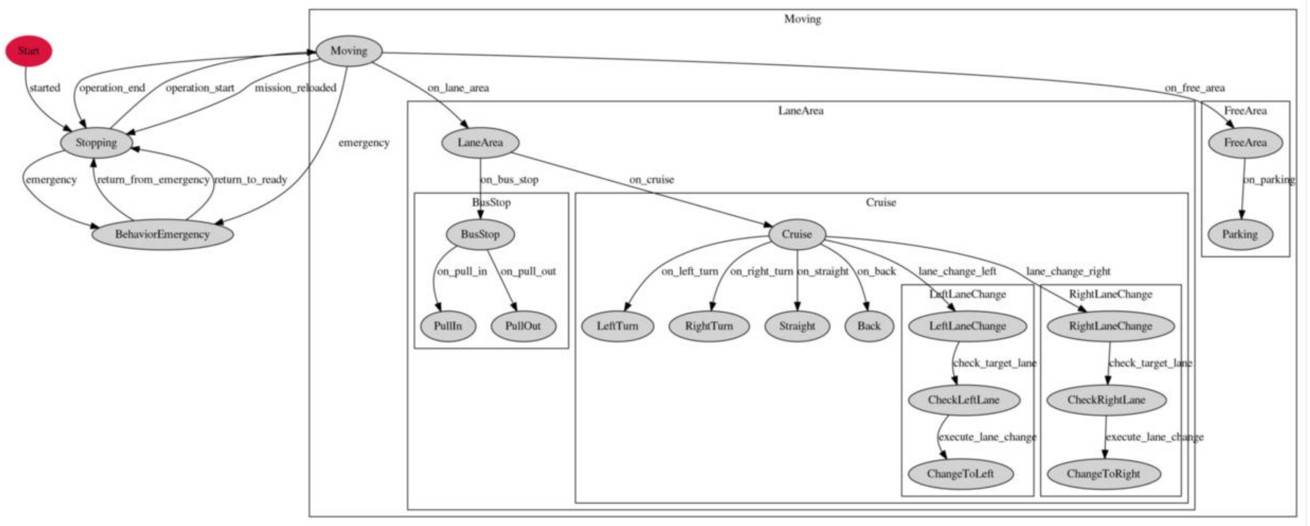


Figure 2.5: 'Decision-maker:Behavior' State-machine Autoware (Source: Github Autoware.io May 2020)

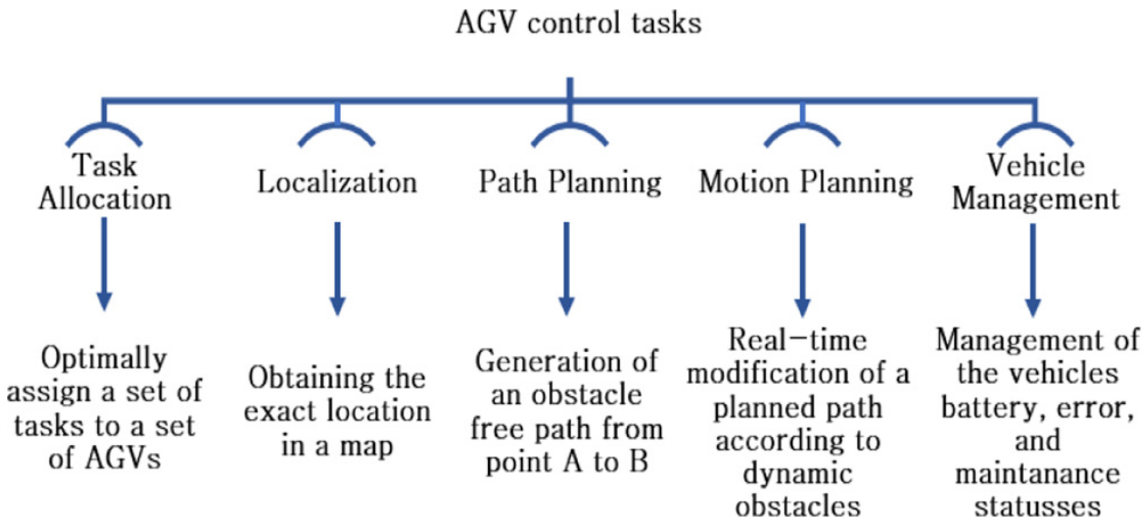

AGV control tasks
Task Allocation
Localization
Path Planning
Motion Planning
Vehicle Management
Optimally assign a set of tasks to a set of AGVs
Obtaining the exact location in a map
Generation of an obstacle free path from point A to B
Real−time modification of a planned path according to dynamic obstacles
Management of the vehicles battery, error, and maintanance statusses

Figure 2.6: Overview of the five AGV Core Tasks (Ryck et al., 2020)

path from start point to goal using known information. During the execution of this path, the motion planner is responsible for modifying the static path based on observations in order to avoid deadlocks due to unforeseen objects. Lastly, monitoring the status of each of the robots is assigned to the *Vehicle Management* task.

Further analysis on the fields of *Path Planning* (Chapter 2.4.3) and *Motion Planning* (Chapter 2.4.4) is done within the defined problem scope of pipe inspection networks.

2.4.1 Centralized or decentralized control

The summary by Ryck et al. (2020) makes an important distinction between a centralized and decentralized control architecture; whether it would be better to have a centralized planner that has control over a single or swarm of robots, or a decentralized approach where each robot is responsible for its own decisions. It is important to make the distinction between these two architectures in order to make a decision for either option for the PIRATE control layer.

As the word implies, a centralized approach to the control and decision making of systems comes down to a single responsible system. This has certain advantages and disadvantages. Ryck et al. (2020) acknowledge that, since this system has all information, path planning for all robots becomes easier, and motion planning becomes trivial. However, this also makes it a single point of failure, and less human-like when the robots are not allowed to use their sensors to self-govern their motion. Liu et al. (2018) actively tested a 'two-layer control structure' consisting of an upper control unit and the lower AGV, which consists of its sensors and some minimal collision avoidance through sonar.

Draganjac et al. (2016) shows a decentralized system in warehouse setting where blocking vehicles negotiate who will halt and allow the other to 'access its private zone'. Fanti et al. (2018) poses a similar approach, where it checks if the pre-defined zones are occupied and, depending on the state of the robots, asks them to relocate, waits, or relocates himself.

These examples show that decentralized approaches often require two important abilities. The robots have to be able to see or know where companions and obstacles are, and they need the physical space to avoid a collision. On top of that, it requires the robot to perform the calculations required to resolve this on its own, which could result in high computational complexity.

2.4.2 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping, or SLAM for short, deals with the orientation and location of the robot, how it perceives its surroundings, and how it stores this information in order to more easily navigate this space after traversing it once. Previous research for the PIRATE with regards to SLAM has already been done. Mennink (2010) designed a SLAM algorithm for the limited amount of sensors that were available on the PIRATE at that time, and more recently Kleiboer (2019), who designed an on-board sensing system for detecting and identifying complex pipe structures. Because of its importance, Vijaykumar (n.d.) will perform a separate research into this topic for the PIRATE robot. Since this work is done simultaneously, it is considered what is required when SLAM is not (yet) available to the robot.

The software already has nodes that deal with camera input, detecting corners and junctions, as Section 2.1.2 explains. Though this certainly does not count as SLAM, it does mean that sensory input that can detect key elements in a pipe network is present. If a map of the network is available, that information can used to construct a crude approximation of the location of the robot.With this approximation it will be possible to create a very basic implementation of localization, in order to proceed with development of the control layer.

2.4.3 Path Planning and Navigation

Path Planning is the method of finding the best path from A to B, a topic in robotics which dates back as early as 1956 by Dijkstra, and is still relevant today, for example in navigation through 3D space. It is analysed how other in-pipe robots and AGVs plan and navigate their environment to gain a better understanding whether such implementations can be used for path planning and navigation for the PIRATE.

According to Le-Anh and Koster (2006), the guide-path problem is the first to consider, which is described as the node-graph topology of Pick-up points, Delivery points, and the aisle intersections. This representation is often used for factory AGV systems. The writers also note that, depending on the complexity of the guide-path, vehicle routing can become quite complex and can easily result in a deadlock of the system.

A more recent example of path planning for AGV systems is the multi-agent scheduling system by Liu et al. (2017). They chose to split the control framework into an upper system, containing the task and scheduling modules, and a lower system, which are the AGVs and their on-board detection systems. The scheduling module is responsible for path planning, which uses a unidirectional direct graph method and the A* algorithm to calculate the route for the AGVs. They conclude that this method produces the optimal path for multiple AGVs (though not the shortest) with regards to high reliability and no collisions due to their 'waiting' strategy.

Research within the research group has been performed towards the planning of AGVs inside an aluminum plant (Bemthuis, 2017). The pathing and planning was outsourced to the openTCS[5] software. This Java application is responsible for Scheduling and Routing the AGVs, which uses an implementation of Dijkstra's with a cost-function for the edges of the graph[6]. It is noted that the default router does not account for expected future behaviour of other vehicles.

These researches apply to 2D scenarios, mostly driving through free open spaces or wider lanes to drive from one *x,y* position to the other *x,y* position. However, the PIRATE robot deals with a different scenario, specifically driving through pipes, where there is no open space that should be navigated, and driving through vertical pipes. The latter means that an ordinary 2D solution is insufficient and a 2.5D (2D with multiple levels) or 3D approach is more suitable to address the pathing and navigation problem for the PIRATE.

[5]https://www.opentcs.org/en/structure.html
[6]https://www.opentcs.org/docs/4.9/user/opentcs-users-guide.html#_default_router

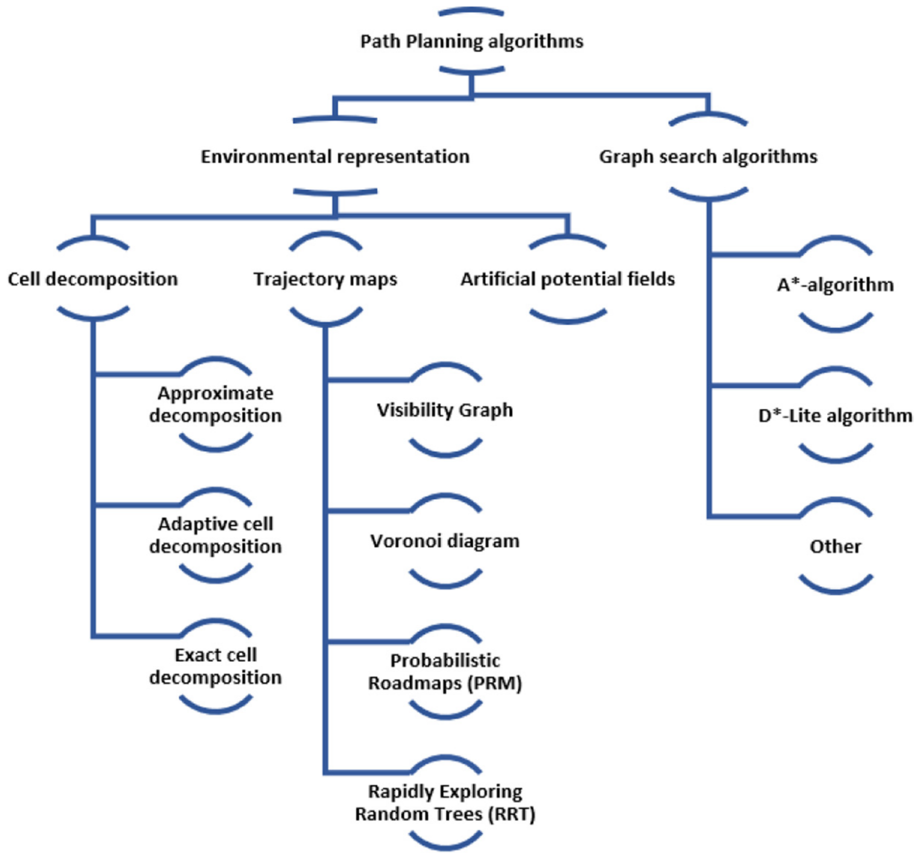

Path Planning algorithms
Environmental representation
Graph search algorithms
Cell decomposition
Trajectory maps
Artificial potential fields
A*-algorithm
D*-Lite algorithm
Other
Approximate decomposition
Adaptive cell decomposition
Exact cell decomposition
Visibility Graph
Voronoi diagram
Probabilistic Roadmaps (PRM)
Rapidly Exploring Random Trees (RRT)

Figure 2.7: Overview of Path Planning Algorithms (Ryck et al., 2020)

Figure 2.7 shows a small summary of different categories path planning algorithms by Ryck et al. (2020). The biggest distinction made here is between the environmental representation and graph search algorithms. The environmental representation describes how the map (Section 2.4.3) is constructed. The graph search algorithms depicted are the most common algorithms to reach the goal (Section 2.4.3).

Environmental Map representation

In order to plan a path from A to B, a location of the starting point and goal is required, for which a map is crucial. Many of the AVG systems use a top-down 2D map of the plant, which can be reduced to a graph consisting of nodes and edges, containing distance or costs. An example of such a 2D petrochemical schematic is given by Dertien (2014) and is shown in Figure 2.8. Since the PIRATE also has to deal with vertical geometries, a 2D map might not look useful at first sight. Nonetheless, whether the map is in 2D, 2.5D, or 3D should eventually not have an effect on the motions given to the robot.

In general, predefined paths are given as a network of nodes, consisting of intersections (nodes) and the paths between the intersections (segments, edges), also known as graphs. (Ryck et al., 2020). When no graph of nodes is available, other methods are used to define the environment, namely cell decomposition which decomposes the environment into a grid of cells with certain sizes, trajectory maps which fill the environment with possible paths that can be taken, or artificial potential fields which defines an attractive force on the goal destination and repulsive forces on objects that should be avoided.

Given the situation that such a map is not available would pose some new questions. For example, what is the goal of the mission, and how can a path be planned to achieve it. This situation may seem unlikely for inspecting pipelines, but might actually be very common due

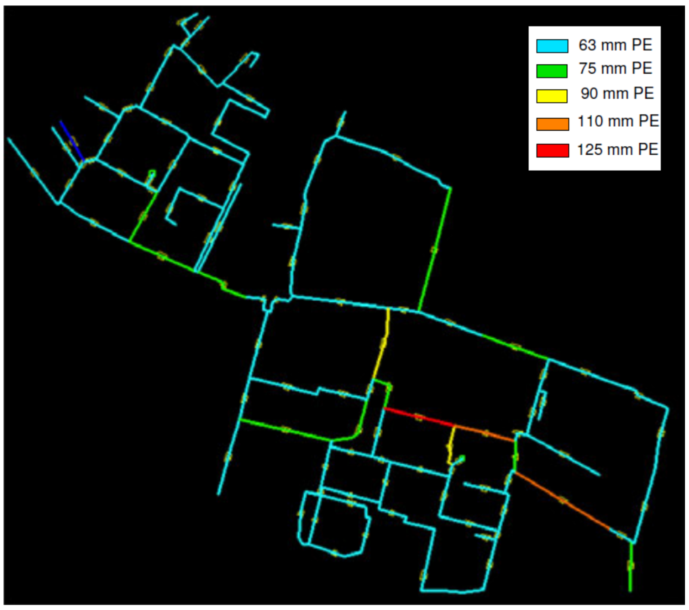


Figure 2.8: Schematic of part of a gas distribution grid in Arnhem, Netherlands. Varying colors indicate the difference in pipe diameter, as indicated by the legend (Dertien, 2014)

to out-of-date maps or no digital representation of them. In this case, path planning as such is not applicable, but an algorithm should still be in control of the robots actions. This means that the control layer must decide which algorithms are used to traverse the network once it is known which information is available, in this case a map, or allow an operator to manually control the mission.

3D Path Planning Algorithms and Graph Search Algorithms

The available path planning algorithms depends on the type of map that is used, as shown in Figure 2.7.

Traversal through graphs is done through Graph Search algorithms, also referred to as Path Finding algorithms, such as Dijkstra, A*, D*, and D*-lite. These algorithms are a means to find a specific path in a graph of nodes and their connecting weighted edges. Given the parameter that should be optimized for, the algorithm will visit some or all of the nodes in order to determine the least costing route under the required circumstances. Depending on the amount of information present in each node or edge, different optimization parameters can be configured. For example, the most common implementation of these algorithms is the A*-algorithm to find the shortest path using the ’length’ of the edges.

In case the map can not be represented as a graph, other algorithms are available to traverse the defined environment. Each of these have their own advantages and shortcomings as Yang et al. (2016) summarizes, but none of them are as optimal as the ’graph search optimal algorithms’.

Operator intervention

Given the fact that no specific use-cases of how the PIRATE will be used are known, it is conceptualised how they could be operated, as this could have an influence on the algorithms chosen, and the design of the decision making process.

From a user perspective it would be optimal to have an overview of the decisions made by the robot, whilst also being able to interrupt and change these decisions when necessary. However, the goal of the PIRATE is to have complete autonomy where an operator would become obsolete. The framework should allow for scaling in this regard, where information about decisions is available, the level autonomy can be configured, which then results in different gradations of autonomy. Frameworks that describe such levels of autonomy are the 'SAE (J3016) Automation Levels' [7] for self-driving vehicles, and Kleiboer (2019) uses the Levels of Robot Autonomy (LORA) framework by Beer et al. (2014), aiming for level "Decision Support".

Examples of such gradations for the PIRATE could be a completely operator controlled mission using only PAB Sequences (J3016 'level 1', or LORA 'Manual'), a mission where the corners a PIRATE wants to make are verified by the operator (J3016 'level 3', or LORA 'Shared Control With Human Initiative'), to an operator-defined mission without any intervention required (J3016 'level 4', or LORA 'Executive Control').

2.4.4 Motion Planning

Motion Planning describes the additional motions required once executing the static path from Path Planning. Unforeseen obstacles should be avoided to ensure that no collisions occur while preventing a deadlock, defined as when the robot has no possible actions to perform (Ryck et al., 2020). Centralized Path Planning has the benefit of having a lot of information about both the network, and location and state of the robots in it, allowing for planning new collision-free paths for the robots at the cost of higher computational complexity, and a single point of failure. A decentralized control architecture allows individual robots to examine the situation based on their own perceptions and act accordingly, resulting in a more robust and scalable solution to the centralized approach. However, this requires more sensors, intelligence, communication between robots, or optimally a combination of the above.

In both cases, it is possible that a robot runs into a dead-end, another vehicle, or gets stuck otherwise. Optimally, individual robots are able to detect these situations and recover from them, whilst simultaneously informing the planner in order to acquire new actions. To support this states such as 'locked' or 'encountering unknown objects' can be used when creating a state machine, similar to the ones created by Autoware as Section 2.4 and Figure 2.5 describes. Recovering from such states requires an intelligent system, like human-operator intervention or a trained system to resolve such deadlocks.

2.5 Simulation environment

In order to speed up the software development process for the PIRATE robot, and due to the global pandemic (Section 1.5), it was decided to test the code in simulation. Even though the simulation will never be a perfect reflection of the actual scenarios, it will help test code quicker, circumvent hardware irregularities that previous researchers encountered with the physical model, and enable testing during the COVID-19 lockdown where the physical model is inaccessible.

An important aspect of the simulation software is that it can run on Linux (due to the ROS requirements), and can deal with ROS messages in order to connect it with the existing PIRATE source code. ROS does not prescribe a simulation platform due to the broad scale of robotics that ROS can be used for. Because of this, many developers find simulation software tailored to their needs, such as Gazebo[8], CARLA[9], and V-REP[10]. The latter, now known as CoppeliaSim, has already been used in previous research for the PIRATE and proven to be feasible for simulating and developing for the PIRATE (Grefte, 2020). Communication between the simulation and ROS is possible through the ROS-bridge[11] and the Python interface PyRep[12], which allows the simulation to send and receive messages based on its sensory input.

[7] https://en.wikipedia.org/wiki/Self-driving_car#Levels_of_driving_automation
[8] http://gazebosim.org/
[9] https://carla.org/
[10] https://www.coppeliarobotics.com/
[11] http://wiki.ros.org/vrep_ros_bridge
[12] https://github.com/stepjam/PyRep

3 Analysis

The following chapter discusses the subjects introduced in Chapter 2 and which systems are required to achieve the design goal. First a decision is made with respect to the separation of systems based on a centralized or decentralized control architecture. Based on this decision, the systems that will be developed are further analyzed, respectively the simulation space and the control layer. Lastly, an analysis is done how these systems interact and how an operator can interact with the newly developed systems through a user interface (UI).

3.1 Separation of systems

Before developing a control layer for the PIRATE, a decision is made for either a centralized or decentralized control architecture as Section 2.4.1 describes.

In order for a decentralized approach to work and have benefit over a centralized approach, two key components are required, respectively that robots can perceive obstacles, and that they can avoid a collision with it in the defined safe regions. However, the PIRATE robot is bound to a pipeline where it has limited line of sight to perceive such obstacles in a timely fashion, and it has zero maneuvering space inside a pipeline apart from reversing or temporarily taking another corner before continuing.

Considering this, our implementation will use a **centralized** control architecture, similar to Liu et al. (2018); the tasks of a robot, and the tasks of planner. Although Ryck et al. (2020) advises against this choice for the sake of future flexibility, it is clear that this design would not fit PIRATE's situation. Developing a centralized system also allows for an easier subdivision of tasks and responsibilities of systems.

The design will consist of an upper system that deals with Path- and Motion Planning, which is the new control layer, and the lower system consisting of the PIRATE robot.

The PIRATE's complexity lies in how it traverses the 3D pipe network and performing inspections. In essence, this is all it should be doing, so the tasks for an individual **PIRATE** are defined as followed:

- Execute a Sequence of motions (e.g. driving, turning, taking corners)
- Report and/or Handle sensor readings
- Perform inspections in the pipe network

The centralized **Mission Planner** control layer is responsible for guiding the robot correctly by providing a sequences of motions so the PIRATE robot can reach the goal destination to perform an inspection. In order to do this, the responsibilities for this control layer inside the scope of this research are defined:

- Receive Map from operator
- Receive Goal from operator
- Calculate Path from position to Goal
- Create List of Motions for robot (e.g. distance to drive, orientation in 3D of corners to take, ...)
- Handle information from external systems in order to update path and list of commands (e.g. user interrupts, information from other robots, etc.)

Due to the centralized approach chosen, each decision is made by the Mission Planner. As soon as the PIRATE's on-board Detection-node defines an object, the PIRATE will inform the Planner, cease any ongoing actions, and await further instructions from the Planner.

3.2 Simulation Environment PIRATE

The PIRATE software is a complex set of systems build in layers by multiple contributors over large spans of time. Due to inaccessibility of the physical robot, a simulation platform is chosen that can couple to the existing source code. Section 2.5 introduces the systems that can be used, and the work done by Grefte (2020) in V-REP. After analysing the proposed simulation platforms, it is concluded that the work done by Grefte (2020) holds the most value with regards to preparing a proper base to start simulations quick and reliably. Based on this, the V-REP simulation software is chosen to be used for testing the control layer on the digital PIRATE model.

Some new systems have to be implemented before V-REP can be used to test the control layer, respectively a control loop, a test environment, and a crude corner detection algorithm.

3.2.1 Control loop

The goal of this research is to create a new control mechanism that leverages all the work that has been done for the PIRATE already. Since this will now be tested in simulation, it is required that the available code base can be used in simulation. This allows for a more realistic representation of the real robot while also using the expertise of the previous research.

This is done through Hardware-in-the-loop (HIL) simulation. In the absence of the real robot hardware, hardware inputs are caught and interpreted by the simulation, which results in outputs from the simulation as if the state of the hardware was changed. This ensures that designed state machines on the 'Loop control' level (Figure 2.1) are used for simulation control.

3.2.2 Pipe Construction

With the control mechanisms in place, the robot can drive, turn, and perform all of its PAB Sequences. To test these, a 3D pipe network is constructed consisting of Figure 2.3 elements (A) horizontal, (D) vertical, (E) elbow bends, and (F) T-sections. The combination of these elements should allow for large networks that test the 3D navigation capabilities of the control layer.

3.2.3 Corner Detection

In order for the PIRATE to drive autonomously through the network, pipe sections of interest like the elbow bend and T-junction have to be properly detected.

Although recent research for the PIRATE has been targeted at the detection and identification of corners (Kleiboer, 2019), no code is available during this research that uses this work to do corner detection based on these methods. A simulated stand-in is developed in order to test the complete autonomous information flow.

3.3 Control Layer Path Planning

The control layer is responsible for navigation and motion, either manually using commands from the operator, or autonomously using a map as representation of the environment, a defined goal objective to reach, and a path finding algorithm that describes the route.

3.3.1 Mission Goal

The mission goal would be defined as what the robot wants to do, which could be a location where an inspection is needed in the network, but might also mean that a robot is sent out to explore the map, or to inspect the entire network. Each of these options will also, similar to the information about the map, have an influence on the path of the robot. It is assumed that the robot is present in the network for a reason, so the mission cannot be null.

Whether the goal is given as a location in 3D, a list of corners to take, or some other form, should all result to the same set of actions and waypoints that are given to the robot.

3.3.2 Environmental representation of the map

In order to plan a path through known pipe network, a representation of the environment is required, a map. The map of this 3D environment should provide the path planning algorithm with enough information such that a path and the required PIRATE motions can be determined from any start position to goal destination.

There are several intricacies that differentiate this structure from a regular maps. The most important differences are connected to the ability to detect obstacles and maneuver in the space, which are actually not mapping problems. However, due to the nature of the pipe network that is to be traversed and the minimal amount of data that a PIRATE robot can acquire while driving through it, it is unnecessary to create a complex map of the network.

Due to the closed pipes, the PIRATE is not required to create a 3D map like the popular 3D point cloud, or an Octomap (based on Octrees)[1] as these will only map the insides of the pipe. A simpler 2.5D representation using *x,y,z* coordinates, or a Multilevel Surface Map as proposed by Triebel et al. (2006) is more suitable for navigating the PIRATE through a pipe network.

Most importantly, the pipes that the PIRATE traverses can be simplified from a three dimensional space to a lower dimensional representation of the landmarks (elbow bends and T-junctions) and the edges between them. Using this perspective, existing and new pipe networks are described through a graph-node network.

However, the map must also provide information about movement and direction so that the PIRATE can be guided through the network upon reaching landmarks. The required information to define the motions between the known nodes is analyzed next.

Edges between landmarks

A simple node-traversal algorithm can move through a node-graph from any defined starting point to any defined destination. However, this does not describe the motions that the PIRATE should go through, as this system does not describe where the next nodes in the graph are, e.g. when the PIRATE reaches landmark A and we want to travel to landmark D, it is not known where this landmark exists in 3D space, nor how to get here.

A solution to this problem is to define the edges between each node with spatial information about the landmarks location relative to the other landmark. However, this information and the successive actions the PIRATE takes are very dependant on the orientation, rotation, and direction of the PIRATE as its reaches the landmark. Figure 3.1 visualizes a use case where a cycle graph will result in differing actions for the same landmark based on the PIRATE's actions before reaching the same landmark.

In order to deal with rotation, each corner can be normalized with respect to gravity (the z-axis). This would give a corner-representation as if the PIRATE were a wheeled vehicle, or a snake always with its belly down. This can be achieved by always physically turning the PIRATE belly-down after a corner, or by tracking in what state the PIRATE is and normalizing the next landmark based on the rotation (face down, left, right, or upside down). The latter is chosen as to not increase the amount of moves needed to traverse between landmarks. This belly-down perspective is dubbed the SNAKE perspective, which would be similar to how a snake or wheeled vehicle would receive its actions. Using a simple state machine, the SNAKE actions can be converted to PIRATE actions.

[1] `https://octomap.github.io/`

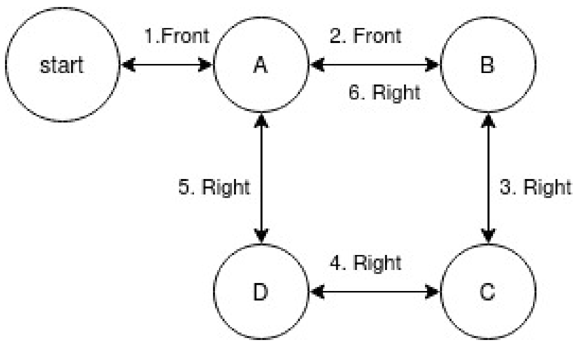


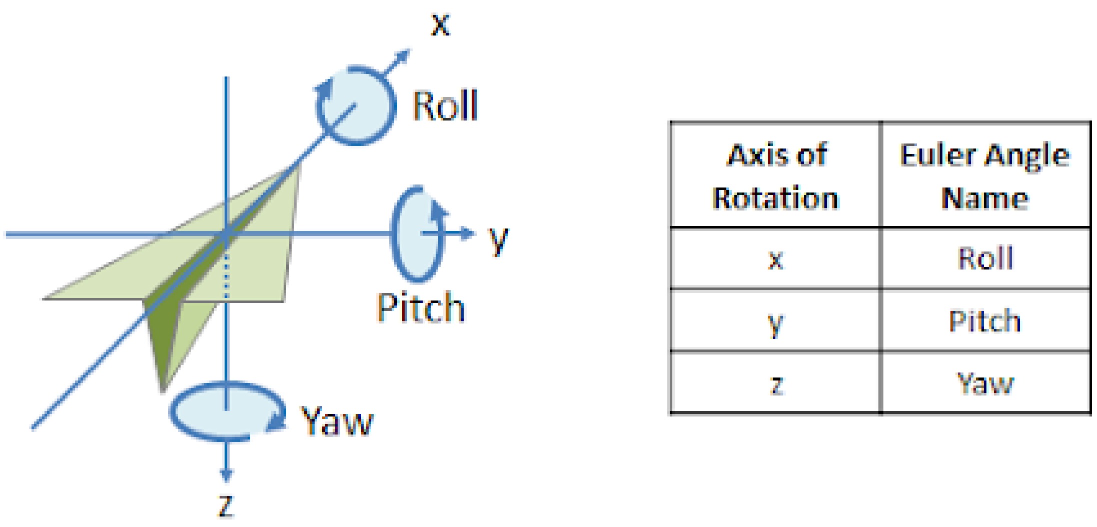
Figure 3.1: The directionality problem between spatial nodes. The path from Node (A) to Node (B) requires a different move when coming from node (start) or node (D).

It can be said that the SLAM algorithm, an existing map, or some other algorithm will result in some spatial 3D location of the identified landmarks. Given these landmark coordinates and some world frame about what these coordinates mean, the positions of the robot relative to the landmarks can be inferred in order to determine which action is required, as the next Section (3.3.2) explains.

Simplifying the Navigation problem (with spatial parameters)

In our structured pipe network, we reach a landmark (crossing) after driving from a previous landmark through a straight pipe, and our destination is another landmark that we reach through a straight pipe from that location. This 3D navigation problem should result in a single variable; the amount of degrees that the PIRATE should rotate in order to be aligned with the direction of the next destination. This is accomplished using vector calculus, a frame to calculate with, and some design decisions, which this chapter explains.

We interpret the straights between these landmarks as vectors, respectively from the previous to current landmark, and current to destination landmark, where our current position is where these vectors are connected in the origin. A 3D representation of rotation is the concept of Roll, Pitch, and Yaw. When we calculate the angle between two 3D coordinates, we speak of yaw when we calculate the angle between the x- and y-plane, of pitch when we calculate the angle between this xy-plane and the z-plane, and of roll as the rotation around the vehicles front-facing axis. This is visualized in Figure 3.2.



Axis of Rotation	Euler Angle Name
x	Roll
y	Pitch
z	Yaw

Figure 3.2: Visualization of Roll, Pitch, and Yaw rotation

Since the Sequence layer only deals with alignments in absolute left and right positions (90 degrees), a test environment is designed that only deals with 90 degree corners as well. This means that we can simplify this calculation for our test case, where we can calculate the yaw and pitch between the two vectors given the world frame that our xyz positions exist in.

Given two 3D coordinates we first calculate the yaw, which is the angle in the xy plane between the two vectors *v1* and *v2*. This is calculated using equation 3.1 which uses the 2-argument arctangent to calculate the angle in Euclidean space, and is taken over the determinant of *v1*,*v2* and the dotproduct of *v1*,*v2* using the x and y components of the 3D vectors. The same equation is used to calculate the pitch between the xy-plane and the z-plane. This equation results in an angle between $-\pi$ and π radians which is used to represent the direction given the frame of the map (e.g. $-\pi$ is Left, π is Right).

$$angle = \arctan 2((v1_x \times v2_y - v1_y \times v2_x), (v1_x \times v2_x + v1_y \times v2_y)) \tag{3.1}$$

Now that we have a direction in two variables, we combine these into a single Roll variable. This is done by looking at which of these two variables describes the most change in direction. If neither of the variables describe big variance of direction, we know that we can go straight forward. If the yaw shows significant change, we align the PIRATE to the right or left, plus some extra rotation for any variability in pitch. If the pitch shows more significant change, we align the PIRATE up or down, plus some extra rotation for any variability in yaw.

3.3.3 Path Finding algorithm

Given the environmental representation of the pipe network as described in Section 3.3.2, a path finding algorithm is selected to best suit this map. Since the map holds information that can determine the motions between landmarks as Section 3.3.2 describes, the only thing path finding has to do is find the 'best' path in the node-graph network.

Existing libraries from ROS are not applicable to this problem, as the 'ROS Navigation stack' only handles 2D navigation, and 'ROS 3D Navigation' (Hornung et al., 2012) only works for Diamondback and later versions (which ROS Kinetic, our environment, is not). Because of this, a custom path planning algorithm is implemented. A common implementation of the 'graph search optimal algorithms' (Yang et al., 2016; Sánchez-López et al., 2019) is the A* path finding algorithm, or a variation of it, i.e. D* or D*-lite.

The A* algorithm is implemented to find the shortest path in the node-graph network. Additionally, using motion information from the map during planning is used to find the 'best' path with respect to the motions of the PIRATE, minimising the amount of turns needed to reach the goal destination. In terms of the PIRATE, this results in the quickest path as turning procedures and corner taking procedures taking longer than driving.

The A*-algorithm

The A* shortest path algorithm requires two sets of the nodes in the network which are 'open' and 'closed', and a cost function when travelling between nodes.

At initialization, the starting node is defined and all sums are set to zero. From this first 'open' node the cost of visiting all other open nodes that can be reached directly from this node (children) given equation 3.2 is evaluated, where n is a child node, $f(n)$ is the accumulated cost from start to n, $g(n)$ is the cost-function up until this node, and $h(n)$ is heuristic function that estimates how useful it will be to visit n to reach the goal as quickly as possible.

$$f(n) = g(n) + h(n) \tag{3.2}$$

Each calculated cost is stored and the nodes are added to the open list. Then the lowest cost open node from the list is picked, moving it to the closed node list, and repeats the process (Equation 3.2) for all its children nodes. This ensures a valid path, costing the same or less than all other available paths to the goal destination.

Section 4.5 describes the calculations used for the quickest path A* cost function using the developed motion descriptions from the map.

3.3.4 User Interface

The current PIRATE setup lacks a user interface that allows for actuating the robot. The proposed solution includes some method for an operator to define missions objectives on a map, take 'manual' control (on the PAB or Sequence level), and intervene with the autonomous decisions made by the system.

To provide these to the operator, a simple user interface is build in the form a terminal. It will allow the operator to manually drive a connected PIRATE robot by sending Sequence commands. It also allows for storing the environmental representation of the network, the map. With this map loaded in, the operator can choose to define missions from the starting location of the robot to any goal destination. Finally, the operator can select to use either the shortest path or quickest path to the goal.

3.4 Approach

The initial version of the complete design is shown in Figure 3.3. This depicts the separation of the PIRATE robot, which jobs starts with the local detecting of objects, and ends with a movement being called at the Sequence node.

Once an object (for now this implies a corner segment) has been detected, the robot informs the Planner and awaits its next command. The Planner, starting with the GuidePathTracker, has a mission definition in the form of waypoints, so it knows which action that PIRATE should perform next. These waypoints will have been defined by the operator or using a map in combination with a path planning algorithm. If localization and a map are present, it can be verified that the robot is indeed at that location in case another action is required.

Once the GuidePathTracker has decided which action should be performed next, the GuidePathDecision class analyses the motion, which concludes whether this decision is correct through motion planning, verifying that such a decision does not result in collisions. Optionally, this is also where the operator is informed and can interfere with the decision made.

If all these steps yield positive results, the GuidePathDecision will send the next motion through a Sequence command to the PIRATE, which can then resume its path until the next detection.

The user interface will be used to input the required settings for the classes in the control layer, visualize the incoming SegmentDetection message, deal with the ExternalConfirmation that is required from an operator, and display any status messages available from the classes in the control layer.

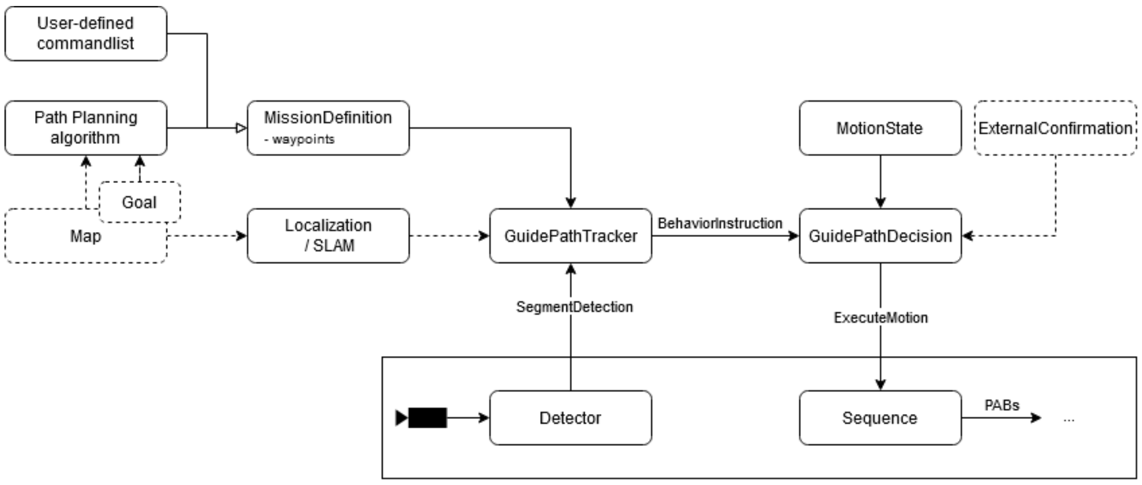


Figure 3.3: Initial design for Information flow of the Control Layer. A detection by the PIRATE is sent to the control layer, which has waypoints defined by a user, or using a path planning algorithm in case a map and goal are defined. The resulting behaviour is converted to a motion and returned to the PIRATE Sequence node.

3.5 Conclusion

After analysis of the required components it is concluded that two new systems have to be created consisting of five components.

The newly developed control layer will be tested in the V-REP testing simulation. To start testing in this environment, a (1) control loop is required to connect the existing PIRATE source code to the simulation model. The simulation will also require (2) a test pipe for the PIRATE to drive through. Since no corner detection is in place in the source code for the PIRATE, (3) corner detection is simulated using information from the simulated environment.

The control layer is responsible for interpreting the information produced by the PIRATE in order to navigate it through the network. In order to do this, the control layer will have a (4) user interface with the environmental representation of the network that includes spatial information of the landmarks. Using the (5) A* path finding algorithm, paths can be calculated from any starting location to any goal location. The path is converted into a set of motions for each landmark that the PIRATE encounters while traversing the network.

4 Design

This chapters describes the design of the five components that were created.

4.1 Hardware-in-the-loop Simulation

The first step is to create a data loop between the existing PIRATE code, written as several ROS nodes that communicate with each other, and the PIRATE simulation model in V-REP.

Controlling the simulation is done through PyRep, a Python library that can control elements in the V-REP simulation. This allows for modifying the actuators of each joint during simulation. This is combined with a Python script that will listen to the messages that are sent to the hardware from the PIRATE code, which results in a good approximation of the behaviour of the physical PIRATE robot.

Figure 4.1 depicts how the existing PIRATE code and simulation are connected.

The hardware simulation consists of two main elements, respectively the *'Movement-Subscriber'* and the *'State-Publisher'*. The *'Movement-Subscriber'* listens to the 'Movement' messages from the *pirate_mov* node that are on their way to actuate the hardware. Since these messages are broadcast in the ROS architecture, we can listen in on these messages and use them to control the robot in simulation. The *'State-Publisher'* is responsible for sending back the state of the simulation as if it were the physical robot to the *pirate_os* class. These systems are described more in-depth in Appendix A.

The simulation is configured using Position control, which causes some problems with the existing TakeTurn Sequence of the PIRATE. Due to translation problems between the Position-control configured simulation model and the Torque-control PIRATE codebase, the TakeTurn Sequence can lead to critical failures in simulation. It is investigated whether calling different PABs would give better results, but as these layers of code were designed to only give very coarse-grain movement options, we did not succeed on this level. Needing more fine-grain tuning, it was analysed whether calling other states or implementing new states into the existing state machine would help with how the simulation would respond to the received commands. These proved to be hard to develop and very time consuming, thus it was chosen to look for other options.

To circumvent the problem and continue the development of the autonomous planner, several alternatives to surpass this process in simulation were looked into. These include the original Plan B of using a different type of robot (like a drone), teleporting the PIRATE to the new pipe location as if the corner was taken, or doing the corner taking using custom simulation-written code. The latter was chosen, as going for Plan B at this stage would be too late, and teleporting would be very unrealistic, where a simulation-driven corner-taking-sequence is closer to the real-world situation.

Figure 4.2 describes a set of motions for the PIRATE to take a T-junction as defined by Dertien (2014). A separate script is written that actuates the simulated joints and wheels through these 11 steps to perform the T-junction corner in simulation.

4.2 Pipe building

In order to test the control of the robot, a test environment pipe structure is required. Based on an early version of the work by Grefte (2020), a pipe structure can be constructed consisting of straight sections and corners. Figure 4.3 shows a pipe structure used for early testing consisting of 6 corners requiring different directions of turning.

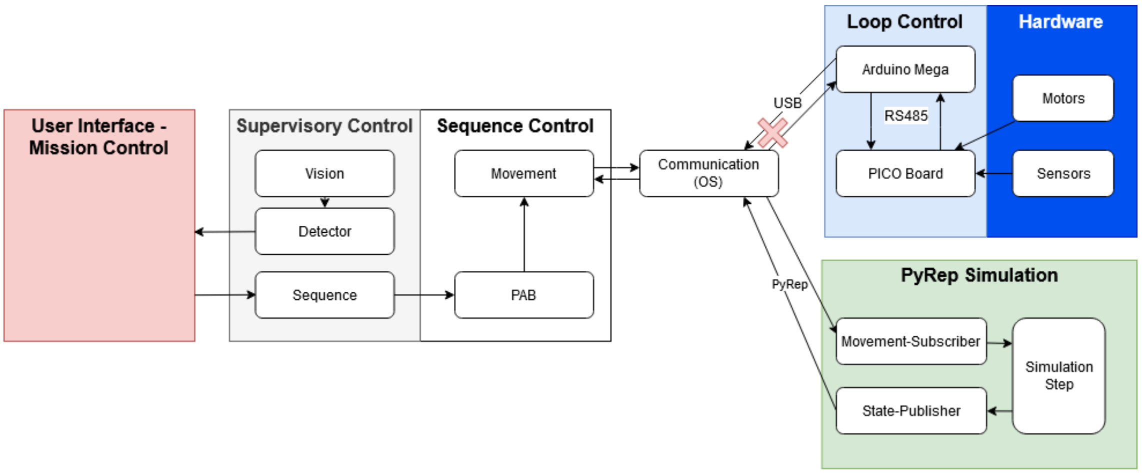


Figure 4.1: Visual representation of the connection between the existing PIRATE codebase and simulation. MOV-Movement messages going to hardware are interpreted in simulation. Simulated hardware state is published as OS-State message. Hardware is not used during this research.

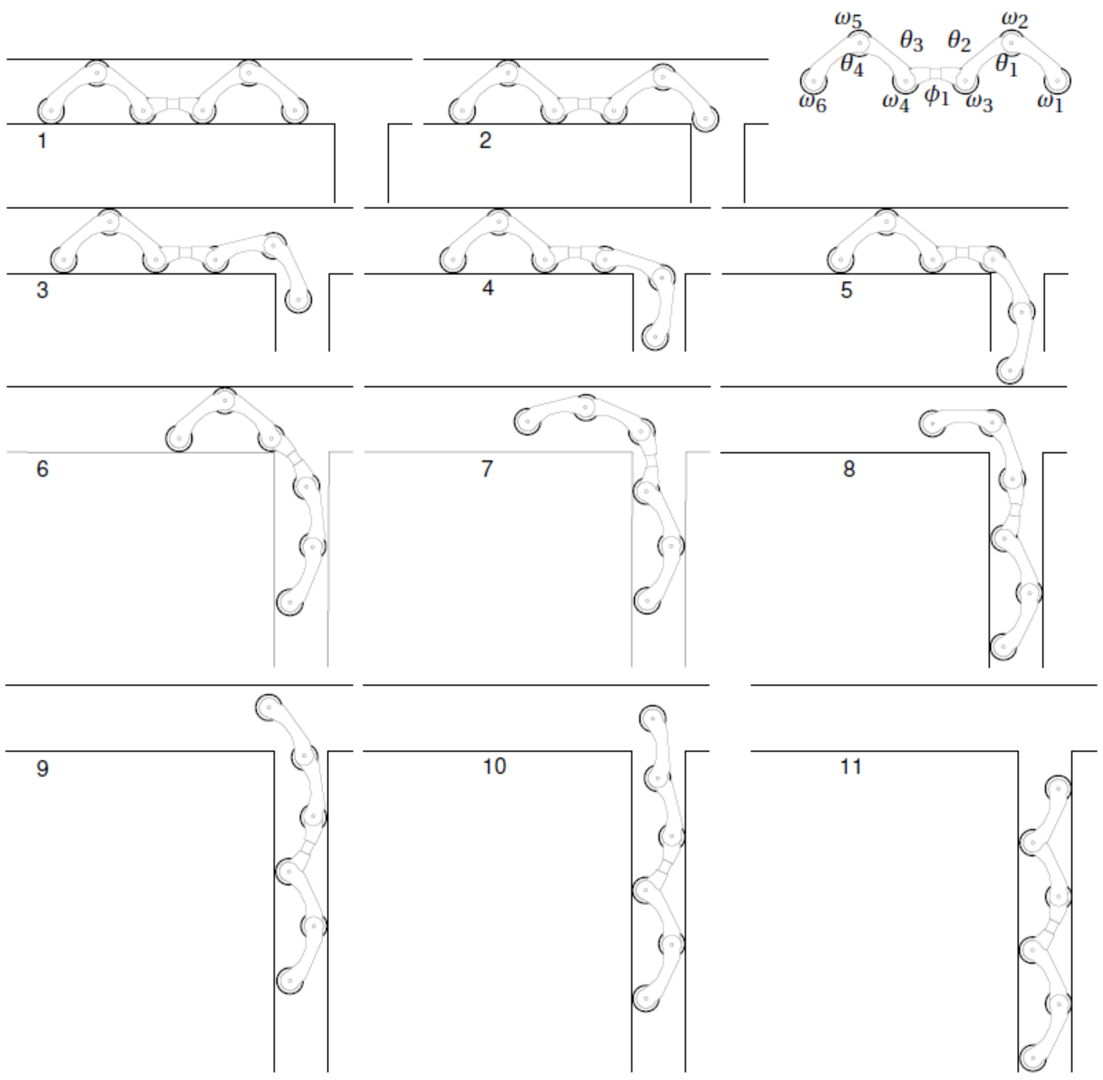
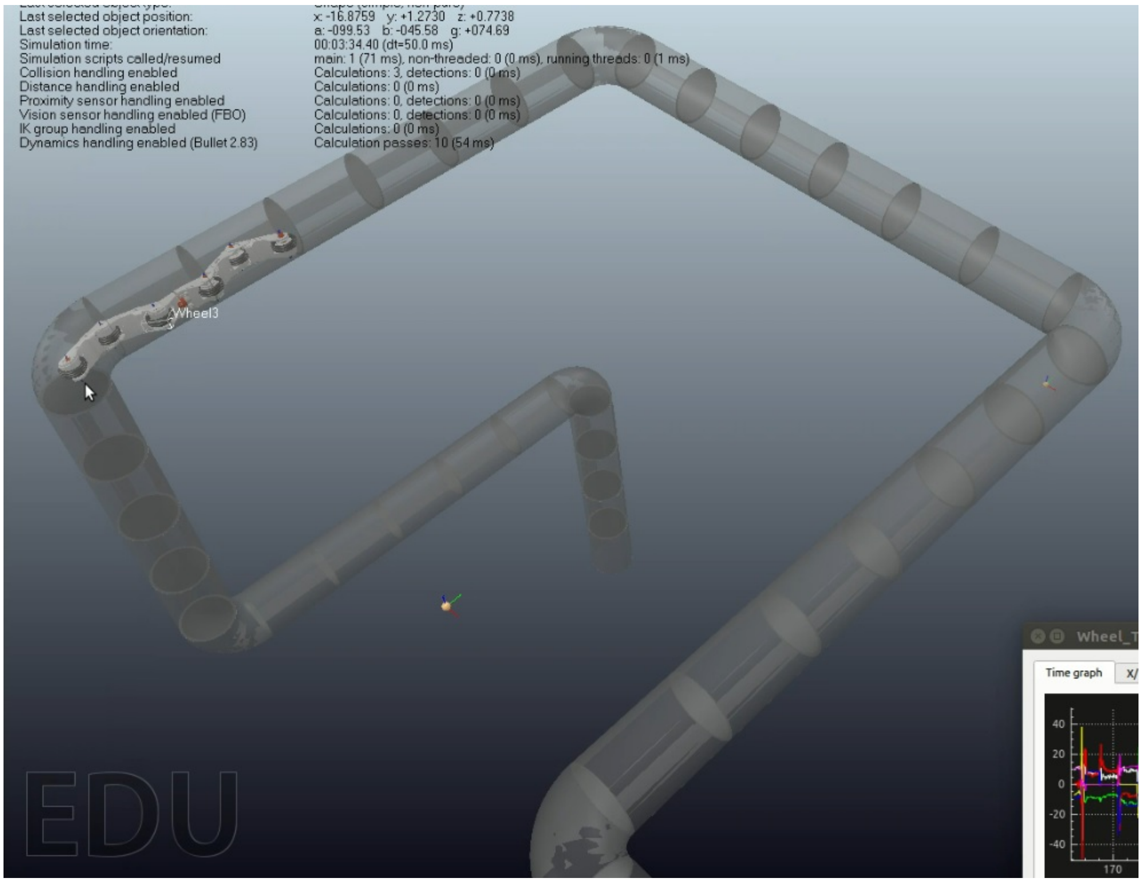


Figure 4.2: Sequence for taking a bend in a T-joint (Dertien, 2014) p.192-193



```
Last selected object position:          x: -16.8759   y: +1.2730   z: +0.7738
Last selected object orientation:       a: -099.53   b: -045.58   g: +074.69
Simulation time:                        00:03:34.40 (dt=50.0 ms)
Simulation scripts called/resumed       main: 1 (71 ms), non-threaded: 0 (0 ms), running threads: 0 (1 ms)
Collision handling enabled              Calculations: 3, detections: 0 (0 ms)
Distance handling enabled               Calculations: 0 (0 ms)
Proximity sensor handling enabled       Calculations: 0, detections: 0 (0 ms)
Vision sensor handling enabled (FBO)    Calculations: 0, detections: 0 (0 ms)
IK group handling enabled               Calculations: 0 (0 ms)
Dynamics handling enabled (Bullet 2.83) Calculation passes: 10 (54 ms)
```

Figure 4.3: Early test pipe in V-REP simulation

This code uses an array of numbers, where each number represents an orientation of the next corner. This method does not provide the sought after level of control over the pipe building method nor the elements to build a complex structure.

To allow for more complex structures, the T-junction element is introduced. Six positions are defined for the new T-junction, respectively Front&Right, Front&Up, Front&Left, Front&Down, Left&Right, and Up&Down, as Figure 4.4 depicts. As this allows the networks to become way more complex, a new method of pipe building is introduced using a graph of corners and T-junctions connected through straight sections.

4.2.1 The pipe structure used for testing

The final pipe structure constructed is meant to have several challenges of varying complexity with regards to pipe-building and navigation in order to increase complexity by changing destinations or changing behaviour.

Using a combination of corners and T-junctions, the maps introduces smaller and larger cycles, and includes both vertical and horizontal sections. It is important to note that all corners are 90 degrees, so the network does not have to deal with corners of differing sharpness. The resulting map is shown in Figure 4.6.

4.3 Corner Detection

The work done on corner detection by Kleiboer (2019) should allow the PIRATE to measure the pipe properties through the vision detection algorithm. PyRep allows for sending data from the vision sensors in simulation to the ROS nodes which would be necessary to couple these systems. However, this code was not available during the development of this phase, and since

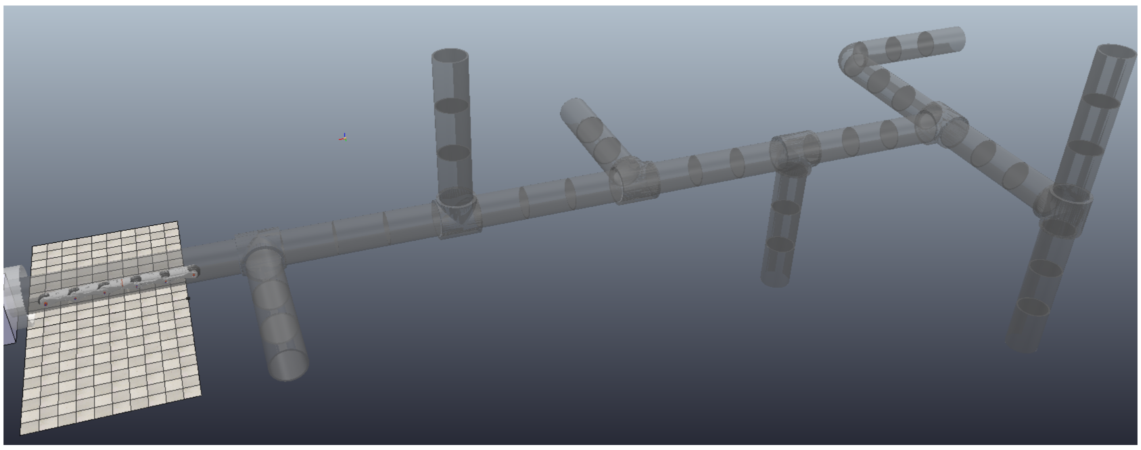


Figure 4.4: The 6 directions in which a T-junction can be placed, respectively *Front&Right, Front&Up, Front&Left, Front&Down, Left&Right,* and *Up&Down*

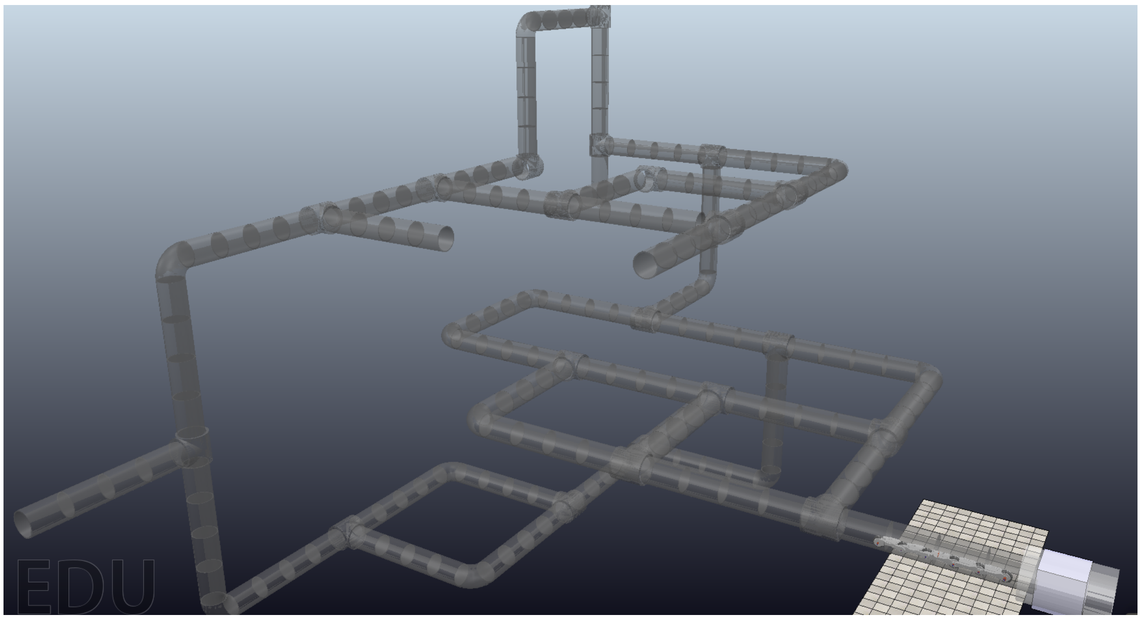


Figure 4.5: Testpipe structure for Iteration 3 in V-REP

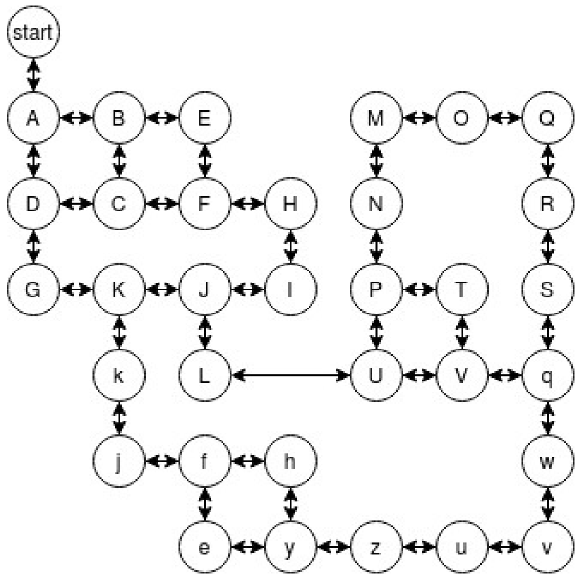


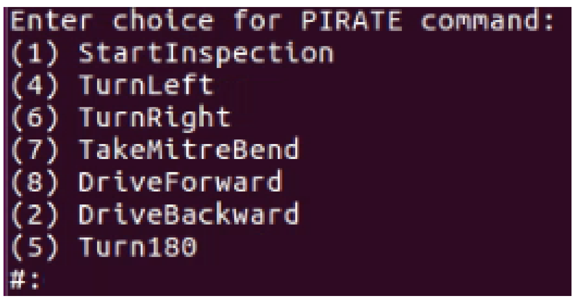
Figure 4.6: Node graph representation of pipe network depicted in Figure 4.5

no previous computer vision node was implemented, an alternative is required. The alternatives that were considered were either through very rudimentary depth analysis from camera input from the simulation, or by automatically 'detecting' the elements when the PIRATE model reaches the corner. The latter is chosen for ease of implementation given that this is not a primary research goal.

The detection is done by creating a list of elements that are bends during the construction of the test pipe. Whenever the PIRATE reaches such an element in simulation, it will brake and will determine the position of the element with respect to the front wheel and rotational module of the PIRATE. These parameters are enough to determine the direction the T-junction or corner is facing, which can then be used to determine how the PIRATE should continue.

4.4 User Interface - Mission Control

The first version of Mission Control is an extra node where an operator can send Sequence commands to the PIRATE, which are existing combinations of the PABs. These commands include, but are not limited to, EnterPipe, AlignLeft, AlignRight, TurnForward, TurnBackward, and three TakeTurn commands. An early version of the user interface for sending these commands is shown in Figure 4.7.

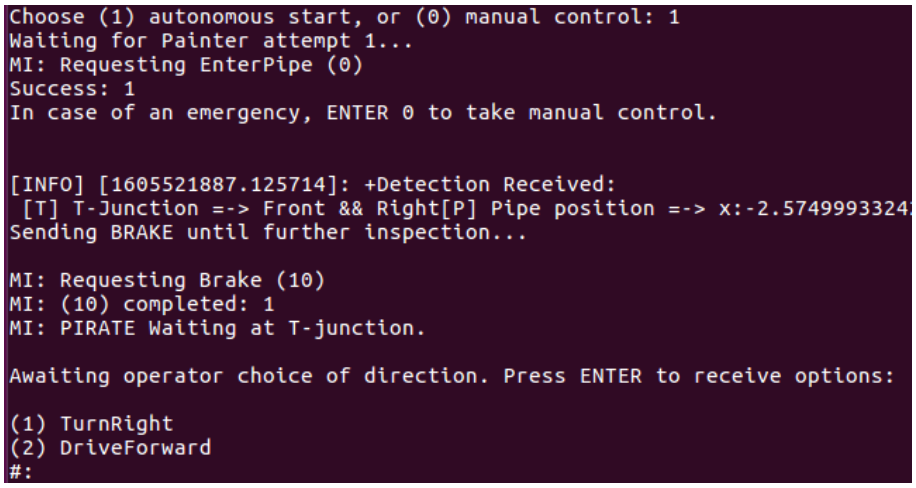


```
Enter choice for PIRATE command:
(1) StartInspection
(4) TurnLeft
(6) TurnRight
(7) TakeMitreBend
(8) DriveForward
(2) DriveBackward
(5) Turn180
#:
```

Figure 4.7: Early version of manual Sequence Control

After introducing the T-junctions and using the corner detection implemented, more detail can be added to the SegmentDetection messages that are sent to the Planner. The new differentiation between corners and T-junctions and the possible routes to take allows us to implement the first level autonomous behaviour to the Mission Planner.

When starting, the operator can choose to control the PIRATE manually through the previously designed Sequence Controller, or start a semi-autonomous mission. This will signal the PIRATE to enter the pipe on its own and start driving until a corner-segment is detected. If this is a regular corner with only one direction to go, the PIRATE will align accordingly and take the corner. If the segment is a T-junction, the PIRATE will brake and await further instructions. The operator will be informed with the available movement options the PIRATE has detected, and once an option has been chosen, the PIRATE will execute the requested Sequence, continuing the mission until a new corner segment is detected. Figure 4.8 shows the described behaviour in the user interface.



```
Choose (1) autonomous start, or (0) manual control: 1
Waiting for Painter attempt 1...
MI: Requesting EnterPipe (0)
Success: 1
In case of an emergency, ENTER 0 to take manual control.


[INFO] [1605521887.125714]: +Detection Received:
 [T] T-Junction =-> Front && Right[P] Pipe position =-> x:-2.5749993324
Sending BRAKE until further inspection...

MI: Requesting Brake (10)
MI: (10) completed: 1
MI: PIRATE Waiting at T-junction.

Awaiting operator choice of direction. Press ENTER to receive options:

(1) TurnRight
(2) DriveForward
#:
```

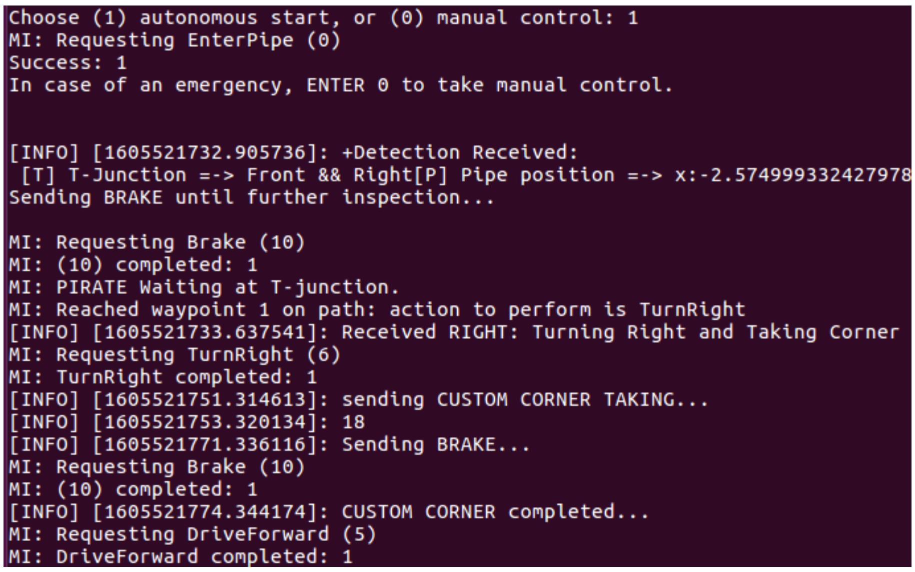
Figure 4.8: Semi-autonomous Mission Control. Given the starting rotation of the PIRATE robot upon reaching landmark 'A', the options at the detected segment are 'DriveForward' or 'TurnRight'.

The last version enables to the operator to start a fully autonomous mission. With the map of the network loaded into the Mission Planner, the operator can select the autonomous mission by defining the starting location, goal destination, and whether the shortest or quickest path is used. After starting the mission, the operator is informed at each landmark of the observations of the PIRATE and the decision that is made to continue traversing that designated path. Figure 4.9 depicts this behaviour. The operator can take back control at any moment, reverting the UI to its semi-autonomous behaviour. After such an intervention, the operator can choose to resume the ongoing mission.

4.5 Path Planning Algorithm

The A* algorithm is implemented as described in Section 3.3.3. First the shortest path implementation is developed. The formula from Equation 3.2 is used. The chosen heuristic function $h(n)$ is the Euclidean 3D distance between node n and the goal node, and the cost-function $g(n)$ is the Euclidean distance between the current node and node n, better described as the length between the current node and n.

In order to plan a more efficient path for the PIRATE, we create a new alternative cost function based on the complexity of moves for the PIRATE, better described as the 'quickest path'. The algorithm starts of the same way as the 'shortest path' implementation, resulting in a set of moves required to reach intermediate landmarks. However, instead of the cost being solely based on the length of the edges and the heuristic function, we apply a new cost function that also gives weight to the intermediate moves required to reach the checkpoint, penalizing 90 degree and 180 degree turns, and reward simpler moves like moving forward. This is done by giving the A* algorithm an extra responsibility to collaborate with the PIRATE motion descriptions. The algorithm has become responsible for remembering the state of the PIRATE at each



```
Choose (1) autonomous start, or (0) manual control: 1
MI: Requesting EnterPipe (0)
Success: 1
In case of an emergency, ENTER 0 to take manual control.


[INFO] [1605521732.905736]: +Detection Received:
 [T] T-Junction =-> Front && Right[P] Pipe position =-> x:-2.574999332427978
Sending BRAKE until further inspection...

MI: Requesting Brake (10)
MI: (10) completed: 1
MI: PIRATE Waiting at T-junction.
MI: Reached waypoint 1 on path: action to perform is TurnRight
[INFO] [1605521733.637541]: Received RIGHT: Turning Right and Taking Corner
MI: Requesting TurnRight (6)
MI: TurnRight completed: 1
[INFO] [1605521751.314613]: sending CUSTOM CORNER TAKING...
[INFO] [1605521753.320134]: 18
[INFO] [1605521771.336116]: Sending BRAKE...
MI: Requesting Brake (10)
MI: (10) completed: 1
[INFO] [1605521774.344174]: CUSTOM CORNER completed...
MI: Requesting DriveForward (5)
MI: DriveForward completed: 1
```

Figure 4.9: Autonomous Mission Control from start to landmark 'V'. Upon reaching landmark 'A', Mission Control executes a TurnRight Sequence and after completion continues to the next landmark.

corner as calculated by the motion functions at each landmark in the path. Once the SNAKE direction of the next move has been calculated, the saved PIRATE state from the previous turn can be used to calculate the PIRATE move that is required to visit node n. Equation 4.1 visualizes this process in pseudo-code.

Each of the six possible motions that the PIRATE can perform at a landmark have been assigned a weight that either penalizes or encourages that move. These are given in Table 4.1. This encourages moving forward through landmarks that require no turning at all. Turning 90 degrees is discouraged over taking a regular corner by a factor of two, and turning 180 degrees by a factor of three.

NextPirateMove	**Weight**
GoStraight	0.5
GoBack	1.0
TakeBend	1.0
TurnLeft	2.0
TurnRight	2.0
Turn180	3.0

Table 4.1: Applied weights to PIRATE motions

$$\begin{aligned} nextPirateMove &= snake2pirate(snakeMove, state[n-1]) \\ g(n) &= g(n-1) + length(n-1, n) \times weight(nextPirateMove) \end{aligned} \tag{4.1}$$

To test if this produces more optimal paths, a test method is developed that counts the combined weights of the produced path.

5 Results

In this research, the newly developed 'Mission' control layer is demonstrated in the existing PIRATE infrastructure, resulting in more autonomous decisions made when controlling the PIRATE, and improving the operator interaction with the PIRATE when no autonomy is possible or selected.

Firstly, a control loop between the existing PIRATE code and the chosen platform of simulation V-REP and PyRep was developed. This makes it possible to use the control mechanisms from previous research in the simulation, while avoiding drawbacks of any technical problems with the physical robot that previous research had to endure. However, the Position-controlled configuration of the simulation had issues with controlling the existing PIRATE code in Torque-control mode, which has led to more work to get the simulation operating. A separate T-junction corner taking sequence for the simulation model enables the simulated PIRATE to take these corners and drive through the pipe network.

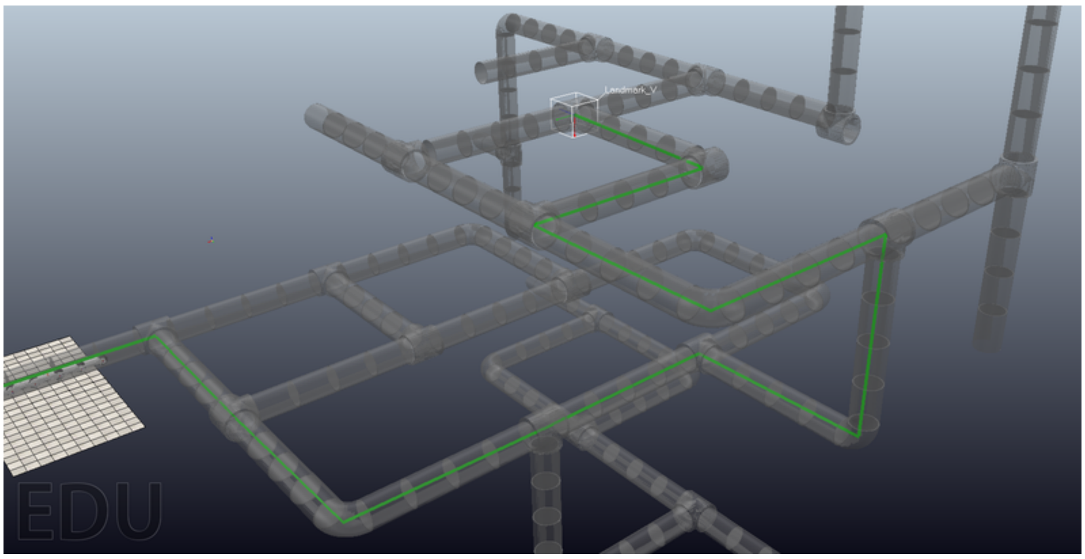
Consequently, the 'Mission' control layer has been constructed, which can be used in several modes. If no map is present, the operator can choose to control the PIRATE manually through Sequences, or semi-autonomous, which enables listening to the landmark-detection messages and giving the operator a choice of direction at each corner. When a map is given, the operator can start an autonomous mission by selecting the path planning settings for shortest or easiest route, followed by a start landmark and a goal landmark. This calculates the path between these two coordinates for the selected setting in both the snake perspective and the PIRATE perspective. Combining this pre-computed set of Sequences with the landmark-detection algorithm results in the PIRATE being led from start to goal through each of the required landmarks.

5.1 A* Path Finding

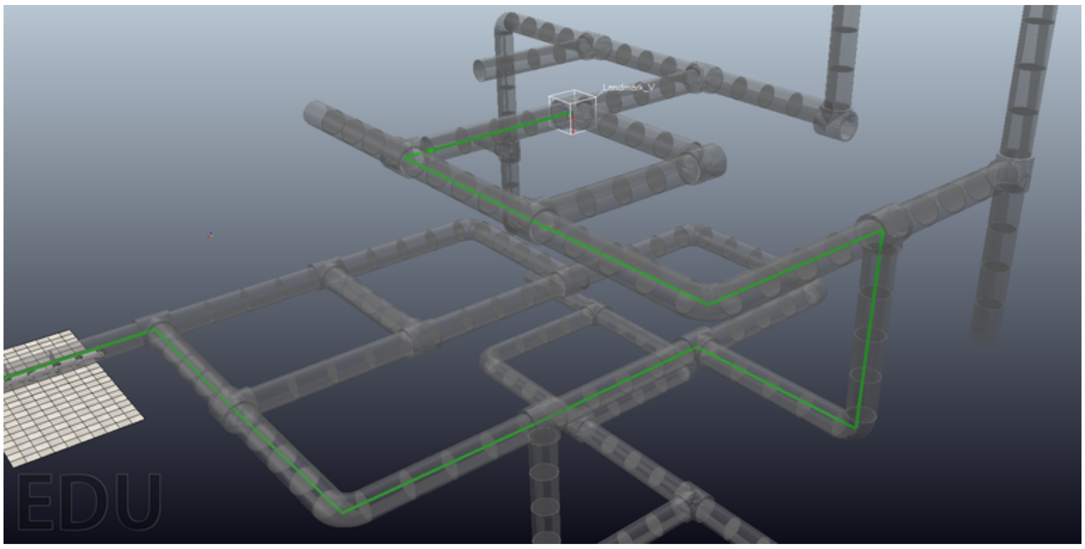
During development two implementations of the A* algorithm were created. Section 3.3.3 describes the general shortest path implementation of the algorithm. Section 4.5 shows the implementation for the quickest path algorithm which uses the available spatial information to determine the path that requires the least amount of turns to the goal destination.

A test case that shows the difference between these two implementations is given in Figure 5.1. A path is requested to landmark 'V', a landmark on the top floor. The shortest path to landmark 'V' results in a path with unnecessary 180 degree turns for the PIRATE as shown in Figure 5.1a. Calculating the same path with the 'quickest' configuration results in a less complex path, as Figure 5.1b depicts.

The complexity of the produced paths is presented in the complexity analysis, which counts the weights of all the required moves for the produced path. Figure 5.2 shows these results for the shortest path to landmark 'V' (Figure 5.2a) and the quickest path to landmark 'V' (Figure 5.2b). This shows that the 'quickest path' algorithm produces a path that requires less complex moves for the PIRATE.

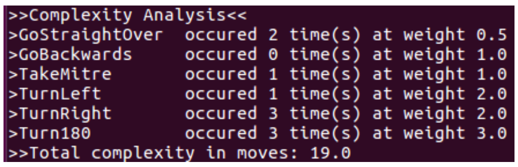


(a) Shortest path to landmark 'V' with an unnecessary 180 degree turn before the end



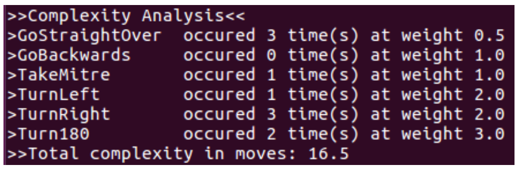
(b) Quickest 'Least Turns' path to landmark 'V'

Figure 5.1: A* algorithm paths to landmark 'V'



```
>>Complexity Analysis<<
>GoStraightOver  occured 2 time(s) at weight 0.5
>GoBackwards     occured 0 time(s) at weight 1.0
>TakeMitre       occured 1 time(s) at weight 1.0
>TurnLeft        occured 1 time(s) at weight 2.0
>TurnRight       occured 3 time(s) at weight 2.0
>Turn180         occured 3 time(s) at weight 3.0
>>Total complexity in moves: 19.0
```

(a) Complexity analysis Shortest path to landmark 'V'



```
>>Complexity Analysis<<
>GoStraightOver  occured 3 time(s) at weight 0.5
>GoBackwards     occured 0 time(s) at weight 1.0
>TakeMitre       occured 1 time(s) at weight 1.0
>TurnLeft        occured 1 time(s) at weight 2.0
>TurnRight       occured 3 time(s) at weight 2.0
>Turn180         occured 2 time(s) at weight 3.0
>>Total complexity in moves: 16.5
```

(b) Complexity analysis Quickest path to landmark 'V'

Figure 5.2: Complexity analysis of path configurations to landmark 'V'

5.2 Demo integration test

Demo One[1] is a video of a demo run through the previously defined pipe network. After initializing the control loop, the V-REP simulation, and the newly developed Mission Control, the goal of this mission is selected to be landmark ’q’, a T-junction in the back of the second floor of the network. Mission Control calculates the quickest path based on the selected ’Least Turns’ cost function, resulting in the following set of PIRATE moves; [*TurnRight, GoStraightOver, Turn180, GoStraightOver, Turn180, TurnRight, TurnLeft, TurnRight, GoStraightOver, TakeMitre, GoStraightOver*]. A green line is displayed in simulation to visualize this path. An in-depth description and explanation of the key events in the video are given in Appendix B.

This demo showcases the V-REP simulation and the control loop, where the PIRATE robot is controlled through the messages of the existing source code as if it were a physical PIRATE robot, and the autonomous corner detection by the digital PIRATE. It also shows the Mission Control user interface, where an autonomous pre-planned mission is executed through a set of motions that the PIRATE performs at each landmark. Finally, the semi-autonomous Sequence controller is used by the operator when the PIRATE gets stuck, to overcome obstacles that the PIRATE could not do on its own. After these manual interruptions, autonomy is reactivated and the mission is resumed.

[1] https://youtu.be/GvrJ0Zfz_9A

6 Discussion

This chapter will deal with the findings, specifically how these contribute to the goal of increased autonomy, if this is in line with the literature, and what is required from finished research and future research to achieve this. Recommendations for future research and development are given in Section 7.1.

6.1 Corner Detection and Feature Extraction

Corner detection was simulated during this research. It was expected that the existing Detection node would have code to do this based on the work of Kleiboer (2019) and Tersteeg (2020), but this was not the case. Nonetheless, we evaluate their work and how these contribute to an autonomous PIRATE robot.

Table 5.3 from Kleiboer (2019) shows a very limited error value of detection in order to do corner detection and feature extraction accurately, where the allowed sensor orientation offset from the centerline is no more than 4 degrees . However, the PIRATE is currently only controlled in absolute motions like *TurnLeft* or *TurnRight* of 90 degrees, which does not allow for correcting on that scale once it is misaligned. Control over the PIRATE should allow for more fine-grain tuning of the physical state of the robot to overcome such challenges. Nonetheless, the results from Kleiboer (2019) are very useful as a corner detection algorithm and should be made available for the physical and simulated PIRATE, where the following should be taken into account.

Mission control currently expects a corner detection message as a list of options in the 6 directions of freedom, so a combination *Front, Left, Right, Up, Down*, and *Back*, instead of an angle. This means that only a minor conversion is needed from the accurate angle detection produced by the LiDAR algorithm to these directions. The easiest solution is that these angles are sent to Mission control who will be responsible for the conversion from angles to directions. However, since such precise control is not available in the Sequence node at this time, it will be challenging to keep the sensor from being misaligned more than 4 degrees.

It should also be noted that Mission control does not take the distance to the detected corner into account, as the simulation would 'detect' the corner at the correct distance. It is important that the information provided by the algorithm with regards to the distance to the corner from Kleiboer (2019) is used in order to start the corner-taking-procedure at the right point in space at the right time.

6.2 Physical control

Since control of the robot in simulation in combination with the existing code through a control loop turned out to be harder than expected, it is also interesting to look at the work done by Grefte (2020) on autonomously driving the PIRATE through Reinforcement Learning, which was recently released from its embargo.

He describes the possibility of using the learned policies in simulation. However, how these could be used for controlling the physical PIRATE is not discussed. Nonetheless, these policies could help with a mixed control scheme for controlling the PIRATE in simulation, where driving and turning is done using the existing code for physical control in combination with the control loop, and then switching to the learned behaviour for more complex tasks like corner taking and diameter changes. However, Grefte (2020) does note that the learned behaviour is very dependant on the model used for learning, including parameters like shape, torque, mass, and friction. Since it is unknown which parameter values were used during learning, it seems unlikely that the results from this work can be incorporated without additional learning required, both for the simulated and physical PIRATE policies. It is worth to note that the same V-REP simulation platform was used during this research.

Nonetheless, the current method of control has some obvious flaws where such learned behaviour could help. The T-junction is traversed using a new script, which is something the physical robot does not currently have access to. More complex pipe segments like cross-junctions, Y-joints, or surpassing valves are currently not described in any control sequence. These challenges could be solved manually in code, but a learned approach might lead to new more error-prone sequences to traverse these elements.

6.3 Path Planning

Path Planning is done with a custom solution rather than using an existing platform or software. Although a lot of algorithms are available for 3D path planning as shown in the summary by Yang et al. (2016), the complexity of these algorithms is often very high, which the pipe network generally is not. We then move to a node based approach, since the pipe network is never more complex than this. Once the path is planned in the node-graph, the next step requires to interpret the 3D coordinates of these nodes and convert these to motions for the robot. Due to information present in the scarce map that is used, considering that no dynamic objects unless otherwise known are present in the map, A* simply is the most optimal algorithm to do this. The A*-algorithm performs as expected in the 3D environment, and correctly finds the shortest path with the normal implementation. The new cost function correctly uses the state of the robot after each move to assess what next motion is required in order to penalize or incentive certain behaviour, resulting in path that requires the least amount of rotations for the PIRATE.

However, the implementation of the A*-algorithm leaves some aspects up for consideration that might be improved upon. One is the usage of Euclidean distance in the heuristic function and angle calculations. The map and pipe looks very similar to what is often used in Manhattan geometry. However, Manhattan distance calculations give almost no benefits over the Euclidean approach, and are less future proof when the network no longer consists only of 90 degree bends, which is why this is not used in this research.

The 3D Euclidean distance is used in both implementations of the A*-algorithm, which can be questioned. In the regular 'shortest path' implementation this is used as the heuristic function, which might not help with looking at more interesting nodes as such. Similar to the A* heuristic in 2D, the algorithm functions almost like a floodfill when a blocking object is encountered near the goal location. In the 3D environment this is even more prominent when a goal location is near the start but is inaccessible without taking a vertical pipe elsewhere in the network. Although this might not be the most optimal heuristic, and other solutions like PRM or other tree-search algorithms as proposed by Yang et al. (2016) might make this process more efficient, the algorithm still produces the correct results, even though more of the network is traversed by the algorithm before finding this solution. For the 'least turns' implementation of the A*-algorithm, the Euclidean distance as a heuristic is not optimal either. The heuristic function should say something about whether node n is a better option than other 'open' child nodes with regards to finding the 'best' path, but the 3D distance to the goal says very little about what turns are made. However, it does incentivize the 'least turns' implementation slightly towards solutions that are in the direction of the goal, which reduces the searched space significantly. Since no other heuristic information is present given the limited amount of data available, this remains to be the best solution.

Finally, this implementation does allow for customization of the weights between nodes in either direction. For example, it is possible to reduce the weight between two landmarks in the vertical downwards direction, increase the weight in the vertical upwards direction, and virtually disable certain pipes by increasing the weight significantly.

6.4 Demo results

The PIRATE has not finished the demo mission shown in Section 5.2. There are several reasons that together have resulted in this outcome.

Firstly, the CAD design of the T-junction is far from optimal due to inexperience of the researcher with the software to create an element that resembles the other elements used in simulation. It has a slightly larger diameter than the pipe straights and bends that are used during construction, which can result in parts of the PIRATE getting stuck, for example when driving or taking a corner. Also, the T-junction design is not perfectly symmetrical, which results in gaps between connected pipe segments, becoming more significant when more distance is between the intended connections. These gaps increase the chance that the PIRATE can not successfully continue the mission after getting stuck. It is not known if this correctly simulates a problem that could occur in a pipe, so whether it is expected that the PIRATE could also overcome such obstacles. If the PIRATE should be able to deal with such pipe structures, additional sensors and interpretation of them is required. The PIRATE simulation does not use any vision or LiDAR system to observe its environment, so it is hard to determine whether it gets stuck. It also does not use any IMU sensor which means that verifying the state and rotation of the robot can not be done.

Lastly, it should be noted that the created script for a taking a T-junction does not produce stable results. A slight difference in distance between the robot and the T-junction, or in rotation of the robot, can result in different outcomes, even when the T-junction is neatly connected in the pipe network. The combination of these variances for each corner after another results in an increased likelihood of a mission failing the further the PIRATE gets.

7 Conclusion

This research has explored what options were available in order to create a system that would allow the PIRATE to perform autonomous missions. The research and development of these systems enables us to answer the posed questions for PIRATE autonomy.

- Which motions can the PIRATE perform and how do these translate to the motions of a typical robot vehicle?

The PIRATE's motion is defined by its PABs and the Sequences that call them. These enable the PIRATE to move forward, backward, turn 90 degrees left and right, start a corner-taking sequence, and brake. However, the frame of the PIRATE changes every time it takes a corner with respect to the world frame. In order to define the motions of the PIRATE while planning a path, these states have to be taken into account. To solve this problem, a motion description is first made for a wheeled vehicle in the world frame. Then, using the behaviour of how the turns affect the state of the PIRATE, a conversion is made from each wheeled action to a PIRATE action. This results in a pre-computed set of motions that the PIRATE should perform using its own frame to navigate to the goal.

- Which input information (sensory and provided data) is minimally required for the control layer to enable a PIRATE to drive autonomously in a pipe network?

In order to autonomously navigate through a pipe structure, this research determined that the PIRATE requires at least some vision-based corner detection and a pre-existing map of the pipe network in the form of 3D landmarks and their edges. Given these two inputs, the newly designed Mission control layer can create a set of actions for the PIRATE that let it traverse from any starting point to any goal.When no corner-detection is in place, the PIRATE is completely dependant on the timing and choices of the motion Sequences sent by the operator. When no map is present but corner-detection is, then Mission Control will inform the operator of the detected pipe segment when one is reached, allowing the operator to take one of the presented choices, or take manual control by sending PAB Sequences. Optimally, SLAM is also present for the PIRATE, which would enable additional features like mapping and localization of the PIRATE inside the pipe network and at the landmarks. Additionally, it would be beneficial if the PIRATE could also acknowledge obstructions while driving or difficulties while taking a corner. Such messages could be delegated to Mission Control in order to calculate a new path to the goal location excluding certain landmarks or edges.

- How can 3D navigation for the physical PIRATE robot be achieved in a pipe network?

PIRATE navigation through a 3D space can be achieved by combining a simple 2D approach with 3D coordinates from a given map. Using a graph-node representation of the map, landmarks are connected to one another and already provide enough information about which landmarks should be traversed to reach any destination. Combining this with 3D information of each of the landmarks allows the path planner to calculate the yaw and pitch between landmarks which can be translated to a single rotational roll angle for the PIRATE to turn in order to become aligned with the next landmark.

Navigation from any starting point to any goal is achieved through a classic A* algorithm to compute the shortest path based on straight edges between each of the landmarks, using a heuristic function based on the 3D locations of the landmarks. Alternatively, the quickest path using a modified A* cost function is calculated which verifies at each landmark which motion is required and, given a weight for this motion, plans a path optimised for using a combination of the least weighted motions.

By answering these questions and designing a new control layer, the design objective was achieved. ***To construct an extra control layer for the PIRATE that deals with navigation and planning to allow for autonomous control.***

PIRATE autonomy is achieved by delegating the decisions at each detected corner to the centralized 'Mission' control layer. A pre-computed path based on 3D landmark node-graph information with an A* path finding algorithm in combination with corner-detection on the PIRATE allows for full autonomy between landmarks inside the pipe network. These motions are first defined for a vehicle that does not change its frame in 2D, like a wheeled vehicle or flying drone. These are then converted to the PIRATE motions for each landmark in the path with regards to the state and frame of the robot at that position.

7.1 Recommendations

To further enhance and improve the autonomy of the PIRATE, it is believed that the following subjects should be focused on in upcoming research.

7.1.1 PIRATE Control

During this research several shortcomings to the existing PIRATE codebase were encountered.

First the PIRATE's model and design has to be finalized, including the required sensors for Corner Detection and Feature Extraction per Kleiboer (2019) and SLAM. This allows for testing the physical robot in networks with the used assumptions, after which the assumptions can be slowly reduced.

Secondly, the PIRATE's control has to be improved. Specifically the amount of fine-grain control in the rotational module has to be increased in order to properly realign the robot with detected corners once misalignment is detected, and new PABs and Sequences have to be developed for more complex corners like the T-junctions as per the design of Dertien (2014).

Finally, if V-REP is used for further digital testing, the choice for configuring the simulation through Position control should be reevaluated. It should be noted that the current T-junction script is written for Position control, so that these cannot be translated one-to-one to a Torque control configuration. Additionally, the pipe building process can be improved by developing a more symmetric T-junction element that reduces the error margin over larger distances.

7.1.2 Corner Detection and Feature Extraction

The work by Kleiboer (2019) and Tersteeg (2020) with respect to corner detection and feature extraction of the important landmarks has proven both useful and necessary for autonomous missions. However, neither codebase were available during this research, for which a simulated work-around was created.

In order to start fully autonomous missions, it is required that these systems are available and work with little error margin. This requires the PIRATE model to first be finalized and include the proper sensor that can perform these scans to properly test these systems on the physical robot.

7.1.3 ROS

It is recommended to move the existing code base from ROS Kinetic, which will reach its End-of-Life (EOL) in April 2021, to the latest version, ROS Noetic (EOL May 2025). This will also provide access to additional libraries that can help with defining world frames and 3D navigation.

Something to consider is moving towards ROS2, although the general opinion as of writing this (late 2020) is that it is not yet ready as a development platform as many of the libraries have not been ported yet. It should be verified that this is still the case when reading this, as it is expected that it will be ready 6 to 12 months from now. ROS2 also allows for better integration with Python3, which is already used in the current PIRATE codebase through the usage of OpenCV4 (with the use of some workarounds to run both simultaneously).

A Simulation Control Loop Message Handlers

The PIRATE simulation loop, as depicted in Figure 4.1, is a means to connect the existing PIRATE C++ source code to the simulated PyRep model. However, the PyRep simulation showed a lot of undefined behaviour when configured through Torque-control (which is how the bend elements are controlled in the existing source code for the real robot). The robot would turn and bend its joints when it should be passive, and motors would respond slow or unexpectedly. The simulation was more stable and showed less unexpected behaviour when using Position control, thus this became the control-method for the simulation.

The 'Movement-Subscriber' listens to the 'Movement' messages that are on their way to actuate the hardware. A MOV message consists of three main elements that have to be decoded before entering the simulation;

- PICO ID
- Motor
- Value

(other values include ControlMode, LimitMin, and LimitMax, but these are generally not set and thus could not be used).

The combination of PICO ID (each module has a PICO board for control) and motor (a module generally has 2 motors, i.e. wheel and bend) describe which wheel, rotational module, bend module, or other peripheral (such as LEDs or camera control) should be actuated. The value parameter describes the amount of actuation. This value is converted to a more suitable representation before being sent to the simulation, i.e. wheel speeds are decreased to simulation appropriate values and sign-flipped for the rear wheels, and rotation and bend values are converted to Euclidean angles and capped when necessary.

The 'State-Publisher' is responsible for sending back the state of the simulation as if it were the physical robot. This is done by taking the actual position of the joints from simulation, converting them back from Euclidean space to the original format, and posting them in the appropriate message slot. This completes the control-loop, which allows code that checks the state of the robot to continue their operations (like the Movement class).

B Text Description of Demo video

The following texts describes in depth what happens at several timesteps during the demo video[1]. The terminal in the top-left shows ’Mission’ control. The rest of the screen is used for the V-REP simulation for testing and visualisation.

00:20: the PIRATE encounters the first landmark (A), a T-junction for which it has to turn right and avoid the other path north of it, which it takes successfully in roughly 45 seconds.

01:10: The second landmark (D) is reached which has to be traversed by first extending the front over the hole (either above or below the PIRATE) and then re-clamping the front and pulling the rear through. For some reason, the PIRATE can not produce enough power with the front to pull the rear through, so manual control is taken at 01:35. A common solution to this simulated problem is turning the PIRATE so it is aligned with gravity in order to make sure the wheels do not slide past the side of the pipe. At 2:05 the PIRATE has passed the landmark, is turned back to its original orientation, after which the autonomous mission is resumed.

02:25: the PIRATE reaches the third landmark (G), a corner which requires it to turn 180 degrees. The PIRATE aligns itself by turning twice, and then successfully takes the corner.

03:30: Landmark four (K) is reached, a T-junction where the PIRATE successfully drives forward without having to deal with any holes above or below it.

04:00: Landmark five (J) is reached, A T-junction that requires a 180 degrees turn. After a minute of trying, the rear has gotten itself stuck as the front could not produce enough power to pull the rear through. To solve this, manual control is taken and the corner-taking procedure is started again, successfully freeing the rear, after which the autonomous mission is resumed.

05:35: Landmark six (L) is encountered several seconds later. This corner is the first vertical section that the PIRATE will have to take to get to the second floor. It is visible that simulated PIRATE has problems taking the corner with the designed procedure, but no critical problems occur due to the simplicity of the bend.

06:25: After driving vertically for several seconds, landmark seven (M) is reached. The PIRATE properly aligns itself for the next corner while staying clamped in the vertical pipe, and then takes the T-junction.

07:10: Landmark eight (N) is reached, a simple corner which is taken with no new problems.

08:00: The PIRATE has reached landmark nine (P), another T-junction that requires the PIRATE to bridge a hole below it. Due to the design of the T-junction model and poor clamping from the rear, the PIRATE’s front section gets stuck on a ridge and fails to reach the other side before clamping. Luckily, nothing critical happens and the PIRATE can be restored by taking manual control, driving the PIRATE backwards, turning him aligned with gravity and driving him through. The PIRATE gets stuck once more on the ridge of the T-junction, which is overcome by stretching the front section forward. Finally, the PIRATE is restored to its previous orientation and the autonomous mission is resumed.

09:50: The PIRATE reaches landmark ten (U), a T-junction where the pipe on the right is slightly uncoupled from the junction. The PIRATE thinks it is properly aligned and immediately starts the turning procedure. However, it is slightly tilted downwards, which in combination with the T-junction model and disconnected pipe to enter, result in the PIRATE having enough space to miss the mouth of the pipe and fold itself into a pretzel under its own force. This critical failure cannot be recovered from, so the mission ends here, two landmarks removed from its destination.

[1]https://youtu.be/GvrJ0Zfz_9A

Bibliography

Beer, J., A. Fisk and W. Rogers (2014), Toward a Framework for Levels of Robot Autonomy in Human-Robot Interaction, *Journal of Human-Robot Interaction*, **vol. 3**, p. 74, doi:10.5898/JHRI.3.2.Beer.

Bemthuis, R. (2017), Development of a planning and control strategy for AGVs in the primary Aluminium industry.
http://essay.utwente.nl/74323/

Broenink, J., M. Groothuis, P. Visser and M. Bezemer (2010), Model-driven robot-software design using template-based target descriptions, in *ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications*, Eds. D. Kubus, K. Nilsson and R. Johansson, IEEE ROBOTICS AND AUTOMATION SOCIETY, pp. 73–77.

Debenest, P., M. Guarnieri and S. Hirose (2014), PipeTron series - Robots for pipe inspection, in *Proceedings of the 2014 3rd International Conference on Applied Robotics for the Power Industry*, pp. 1–6.

Dertien, E. (2006), System specifications for pirate, *Edwin Dertien, Control laboratory, University of Twente, the Netherlands.*

Dertien, E. (2014), *Design of an inspection robot for small diameter gas distribution mains*, Ph.D. thesis, University of Twente, Netherlands, doi:10.3990/1.9789036536813.

Draganjac, I., D. Miklic, Z. Kovacic, G. Vasiljevic and S. Bogdan (2016), Decentralized Control of Multi-AGV Systems in Autonomous Warehousing Applications, *IEEE Transactions on Automation Science and Engineering*, **vol. PP**, pp. 1–15, doi:10.1109/TASE.2016.2603781.

Fanti, M. P., A. M. Mangini, G. Pedroncelli and W. Ukovich (2018), A decentralized control strategy for the coordination of AGV systems, *Control Engineering Practice*, **vol. 70**, pp. 86 – 97, ISSN 0967-0661, doi:https://doi.org/10.1016/j.conengprac.2017.10.001.
http://www.sciencedirect.com/science/article/pii/S0967066117302253

Garza Morales, G. (2016), Increasing the Autonomy of the Pipe Inspection Robot PIRATE.
http://essay.utwente.nl/71300/

Geerlings, N. (2018), Design of a high-level control layer and wheel contact estimation and compensation for the pipe inspection robot PIRATE.
http://resolver.tudelft.nl/uuid:0db67388-371b-4f54-bb3c-b07521043637

Grefte, L. (2020), Autonomous navigation of the PIRATE using Reinforcement Learning.
http://purl.utwente.nl/essays/83150

Hoekstra, G. (2018), Towards a software architecture model for the automation of the PIRATE robot.
http://essay.utwente.nl/74563/

Hornung, A., M. Phillips, E. Gil Jones, M. Bennewitz, M. Likhachev and S. Chitta (2012), Navigation in three-dimensional cluttered environments for mobile manipulation, in *2012 IEEE International Conference on Robotics and Automation*, pp. 423–429.

Kakogawa, A. and S. Ma (2020), Robotic Search and Rescue through In-Pipe Movement, in *Unmanned Robotic Systems and Applications*, Eds. M. Reyhanoglu and G. D. Cubber, IntechOpen, Rijeka, chapter 1, doi:10.5772/intechopen.88414.
https://doi.org/10.5772/intechopen.88414

Kato, S., S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii and T. Azumi (2018), Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems, in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 287–296.

Kleiboer, D. (2019), Towards autonomous motion of the PIRATE through closed pipe structures The Sensing System.
`http://essay.utwente.nl/80280/`

Le-Anh, T. and M. D. Koster (2006), A review of design and control of automated guided vehicle systems, *European Journal of Operational Research*, **vol. 171**, pp. 1 – 23, ISSN 0377-2217, doi:https://doi.org/10.1016/j.ejor.2005.01.036.
`http://www.sciencedirect.com/science/article/pii/S0377221705001840`

Li, Q., J. Udding and A. Pogromski (2010), Modeling and control of the AGV system in an automated container terminal, in *Proceedings of the 30th IASTED Conference on Modelling, Identification, and Control 24-26 November 2010, Phuket, Thailand*, pp. 1–8.

Liu, C., J. Tan, H. Zhao, Y. Li and X. Bai (2017), Path planning and intelligent scheduling of multi-AGV systems in workshop, in *2017 36th Chinese Control Conference (CCC)*, pp. 2735–2739, ISSN 1934-1768, doi:10.23919/ChiCC.2017.8027778.

Liu, J., X. Zhong, X. Peng, J. Tian and C. Zou (2018), Design and Implementation of New AGV System Based on Two-layer Controller Structure, in *2018 37th Chinese Control Conference (CCC)*, pp. 5340–5346.

Mazraeh, A., F. Ismail and A. Alta'ee (2014), RFEC PIG Designed for Long Distance Inspection, cp-395-00049, ISSN 2214-4609, doi:https://doi.org/10.3997/2214-4609-pdb.395.IPTC-17487-MS.
`https://www.earthdoc.org/content/papers/10.3997/2214-4609-pdb.395.IPTC-17487-MS`

Mennink, T. (2010), Self localization of PIRATE inside a partially structured environment.
`http://essay.utwente.nl/69616/`

Mills, G., A. Jackson and R. Richardson (2017), Advances in the Inspection of Unpiggable Pipelines, *Robotics*, **vol. 6**, p. 36, ISSN 2218-6581, doi:10.3390/robotics6040036.
`http://dx.doi.org/10.3390/robotics6040036`

ya Nagase, J., F. Fukunaga, K. Ishida and N. Saga (2018), Steering System of Cylindrical Elastic Crawler Robot, *IEEJ Journal of Industry Applications*, **vol. 7**, pp. 441–442, doi:10.1541/ieejjia.7.441.

Ryck, M. D., M. Versteyhe and F. Debrouwere (2020), Automated guided vehicle systems, state-of-the-art control algorithms and techniques, *Journal of Manufacturing Systems*, **vol. 54**, pp. 152 – 173, ISSN 0278-6125, doi:https://doi.org/10.1016/j.jmsy.2019.12.002.
`http://www.sciencedirect.com/science/article/pii/S0278612519301177`

Sánchez-López, J. L., M. Wang, M. A. Olivares-Méndez, M. Molina and H. Voos (2019), A Real-Time 3D Path Planning Solution for Collision-Free Navigation of Multirotor Aerial Robots in Dynamic Environments, *Journal of Intelligent & Robotic Systems*, **vol. 93**, pp. 33–53.

Schempf, H. and G. Vradis (2003), Explorer: Untethered Real-Time Gas Main Assessment Robot System, in *Proceedings of the 20th International Symposium on Automation and Robotics in Construction ISARC 2003 – The Future Site*, Eds. G. Maas and F. Van Gassel, International Association for Automation and Robotics in Construction (IAARC), Eindhoven, The Netherlands, pp. 471–476, ISBN 978-90-6814-574-8, doi:10.22260/ISARC2003/0074.

Suryavanshi, K., R. Vadapalli, R. Vucha, A. Sarkar and K. M. Krishna (2019), Omnidirectional Tractable Three Module Robot, *CoRR*, **vol. abs/1909.10277**.

http://arxiv.org/abs/1909.10277

Tersteeg, S. (2020), Autonomous driving the pirate robot.
https://essay.utwente.nl/83150/

Triebel, R., P. Pfaff and W. Burgard (2006), Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing, in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2276–2282.

Vijaykumar, S. (n.d.), SLAM Research PIRATE: N/A.

Yan, X., C. Zhang and M. Qi (2017), Multi-AGVs Collision-Avoidance and Deadlock-Control for Item-To-Human Automated Warehouse, in *2017 International Conference on Industrial Engineering, Management Science and Application (ICIMSA)*, pp. 1–5.

Yang, L., J. Qi, D. Song, J. Xiao, J. Han and Y. Xia (2016), Survey of Robot 3D Path Planning Algorithms, *Journal of Control Science and Engineering*, **vol. 2016**, p. 7426913, ISSN 1687-5249, doi:10.1155/2016/7426913.
https://doi.org/10.1155/2016/7426913