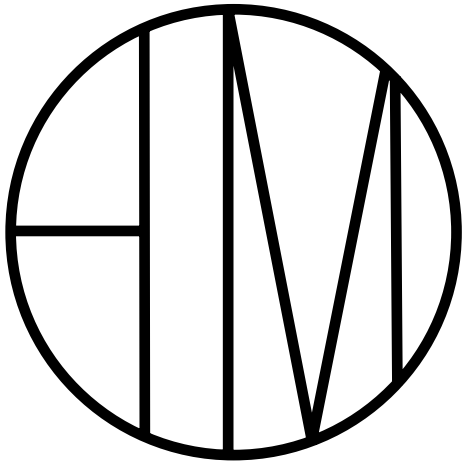
Design Specification
LiU Racetrack 2018

LiU Racetrack 2018
Author: Oskar Karlsson
2018-11-06
Version 1.0



TRUKKMANIA

Status

Reviewed	Oskar Karlsson	2018-11-06
Approved	Oskar Ljungqvist	2018-11-06

Project identity

2018 HT, LiU Racetrack 2018
Department of Electrical Engineering - Linköping University

Name	**Role**	**Email**
Tomas Busk (TB)	Responsible for LEGO Truck	tombu079@student.liu.se
Mergim Dushku (MD)	Responsible for Migration to ROS	merdu044@student.liu.se
David Forsberg (DF)	Responsible for Migration to Visionen	davfo471@student.liu.se
Marcus Hall (MH)	Responsible for Simulation Enviroment	marha585@student.liu.se
Oskar Karlsson (OK)	Document Manager	oskka906@student.liu.se
Kim Larsson (KL)	Software Architect	kimla207@student.liu.se
Jacob Mourad (JM)	Project Leader	jacmo228@student.liu.se
Sara Nilsson (SN)	Test Manager	sarni339@student.liu.se

Orderer: Oskar Ljungqvist, Linköping University, +46 70 577 18 68, oskar.ljungqvist@liu.se
Course responsible: Daniel Axehill, Linköping University, +46 13 28 40 42, daniel.axehill@liu.se
Advisor: Olov Holmer, Linköping University, +46 13 28 16 17, olov.holmer@liu.se

Contents

Document history

Version	**Date**	**Changes**	**Performed by**	**Reviewed by**
0.1	2018-09-28	First draft	Project Group	OK
0.2	2018-10-10	Revised after comments from advisor.	Project Group	OK
0.3	2018-10-18	Revised after additional comments from advisor.	Project Group	JM
0.4	2018-11-05	Revised version after comments from orderer.	Project Group	OK
1.0	2018-11-06	Approved by orderer.	Project Group	OK

1 Introduction

This document is the design specification for the project LiU Racetrack 2018. The project is part of the course *Automatic Control - Project Course* given by the department of Electrical Engineering, ISY, at Linköping University. This project has been developed over several years and has for the last couple of years consisted of a few small cars and a semi-trailer truck. The focus in this year's project is to migrate the system to ROS and integrate a LEGO truck platform into the system.

The long term goal for the project is to create a robust system that can be used both for research and educational purposes. In this year, the main goals are to firstly evaluate the current system while modularizing and migrating it to ROS. Secondly, a new system containing an autonomously driving LEGO truck with a dolly-steered trailer that can perform advanced parking maneuvers will be delivered.

1.1 Definitions

- **Original system** - Refers to the system developed prior to the start of this year's project.
- **Semi-trailer truck** - Refers to the truck from previous years.
- **LEGO truck** - Refers to the new LEGO truck that will be implemented in the system this year.
- **ROS** - Robot Operating System.
- **roscore** - Is the main-program for the ROS setup. This program will run on the *master-node*.
- **Visionen** - Visionen is an area located at Linköping University, often used to present various types of projects. This report will mention *Stora Visionen* and *Lilla Visionen*, which are two different areas that are located in Visionen.
- **Frenet frame** - A moving coordinate system that follows a reference trajectory.

2 System overview

The system will consist of several subsystems. The system with a semi-trailer truck and car can be seen in figure 1.

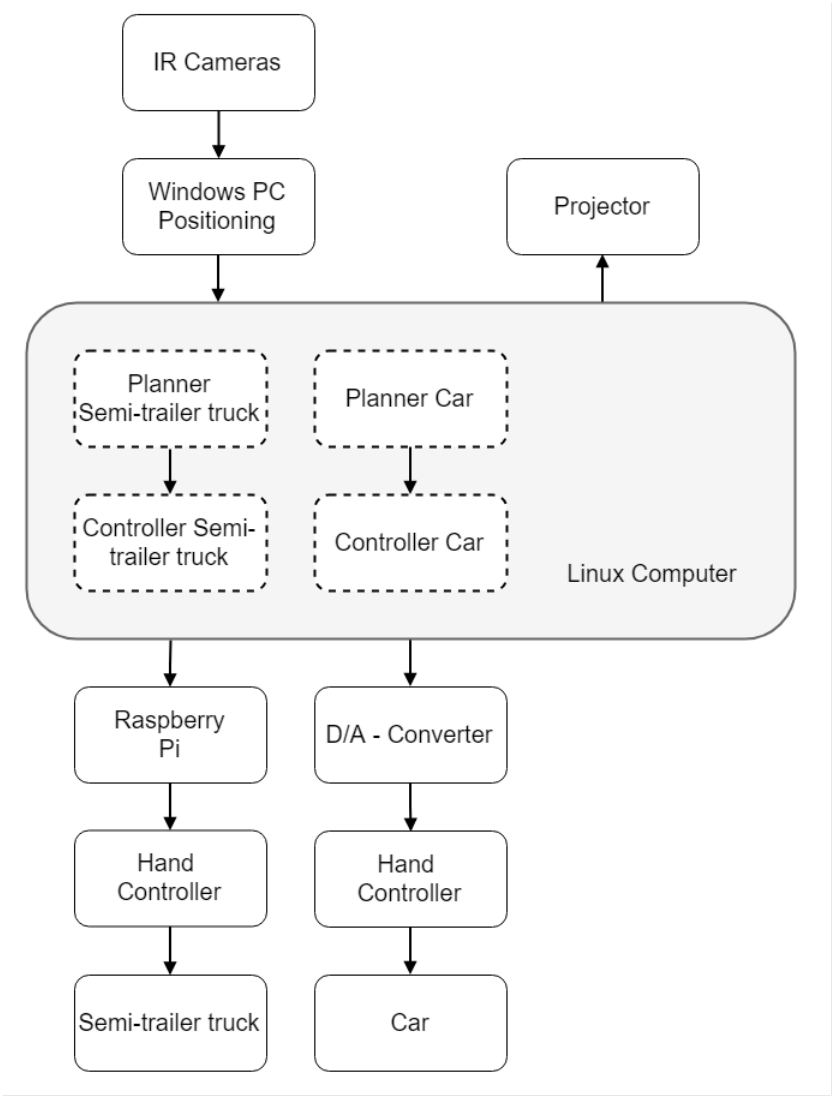


Figure 1: Overview for the system with semi-trailer truck and car.

The system consists of IR cameras that sends their feed to a dedicated Windows PC which estimates the states of each vehicle. This information is then transmitted to a Linux computer where intelligent motion planners generate motion plans for the vehicles. These motion plans together with the current vehicle states are then sent to the vehicle controllers for the car and semi-trailer truck, respectively. The vehicle controllers will then execute the calculated motions plans by assigning appropriated control signals to the vehicles. These control signals are sent to a Raspberry Pi for the semi-trailer and a D/A-converter for the car to their respective hand controller. The hand controllers make the vehicles move by controlling their actuators through a wireless connection.

The other main system is the LEGO truck. This system will use an external positioning system that is intended to provide information regarding the vehicle pose to a Linux computer. On the Linux computer, a motion planner will be developed. The planner will calculate motion plans that will be sent to the LEGO truck for plan execution and to a projector for visualization, see figure 2. A more detailed description of the system is provided in section 6.

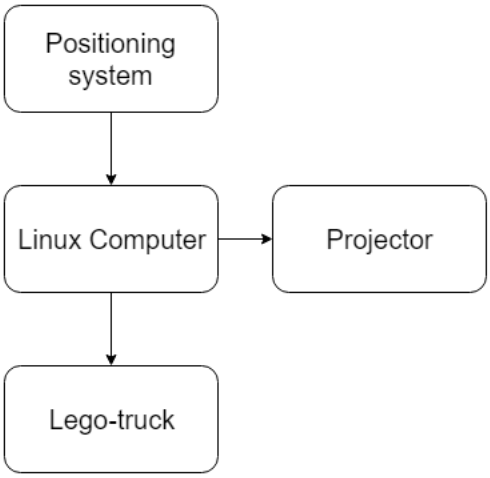


Figure 2: Overview for the system with the LEGO truck.

3 Migration to ROS

The original system will be modularized and reconstructed using ROS. Sections of the original system will be represented as nodes in ROS and it is therefore important to make sure that no functionality is lost during the migration. If some functionality is lost there is a high risk that the system will not function as intended.

The reconstruction will consist of a set of ROS nodes where interfaces between the nodes will be defined. This reconstruction will result in a more flexible and scalable system since a developer may replace nodes as long as the interfaces are the same. An example would be if the system is desired to have a replaceable controller for the car's lateral deviation. This could easily be done if the interfaces are kept the same among the interchangeable controllers. An overview of the ROS system can be seen in figure 3.

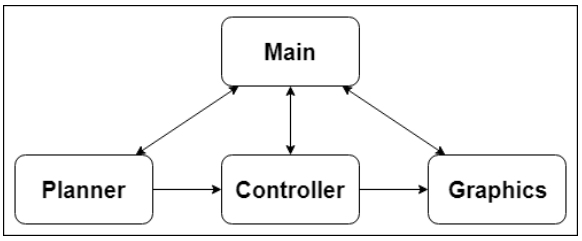


Figure 3: An overview of the ROS implementation.

The design of the ROS nodes will start as a model of the complete system similar to the original system. The original system consists of two threads, *Main* and *Run*, that handles the system's launcher GUI and the starting of lower threads. Our solution will almost entirely remove these threads and instead create a main-node from scratch that uses a ROS tool called *rqt* to set up a launch GUI, figure 4 shows an example of the rqt tool. More information about the rqt tool can be found in [1].

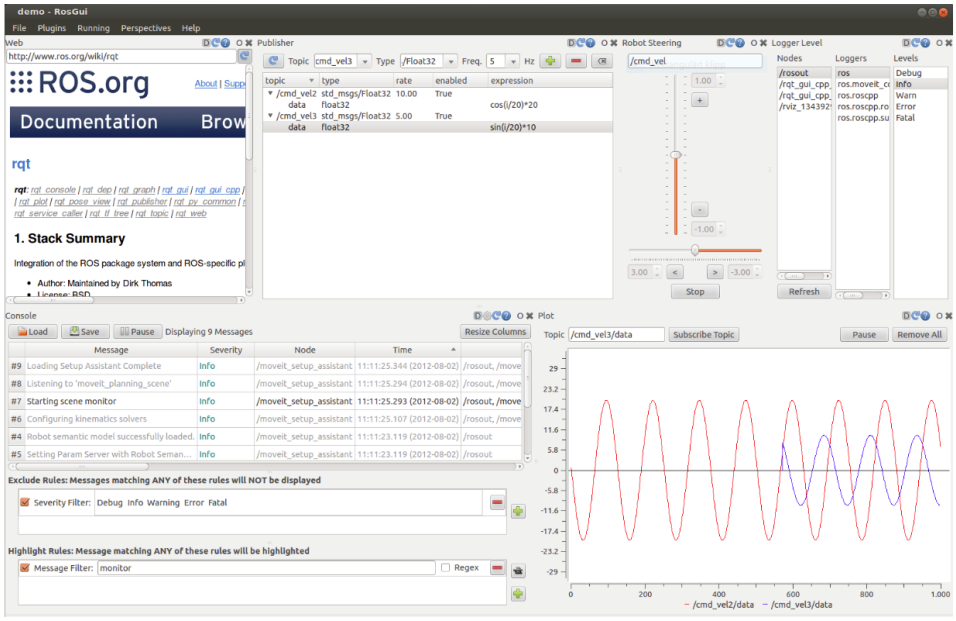


Figure 4: An example of the rqt tool, taken from [1].

The nodes seen in figure 3 will be divided further after a more thorough analysis of the existing code has been made in order to determine the existing underlying software structure. The idea is to try to modularize the nodes as much as possible in order to make the system more scalable and flexible in future projects.

3.1 Challenges

Below are some challenges that the group will have to deal with when implementing the ROS based system.

- Messages between Linux and Windows, since Windows does not originally support ROS.
- Previous solutions might not work in ROS and need to be reevaluated.
- Evaluated parts needed to be completely rewritten.

4 Simulator

The simulator will be a remade version of the existing one from previous year where the behavior will be the same although the simulator itself will be completely remade. The new simulator will be developed using the Gazebo simulator to simulate the different ROS nodes. This will be visualized using the Rviz visualization tool which is tightly connected to ROS. This will be a clear advantage compared to the previous simulator since it will use the ROS nodes from the real program as a base for the simulations.

The purpose of the simulator will be to evaluate the motion planners for the car, the LEGO truck and the semi-trailer truck, respectively. In general, the new simulator is going to have the same behaviour as the original simulator, but since the system is to be migrated to ROS it makes sense to also move the simulator to a more appropriate environment. In addition to the existing functionality in the simulator there will also be an interest in testing behaviour for the parking functions that are to be implemented, explained further in section 7. As a first step, a single map with models for the semi-trailer truck, the car and for the LEGO truck will be implemented. When this is working, and if there is time, several maps will be implemented to choose between as well as the ability for the user to make their own maps.

A plug-in to Gazebo for the LEGO truck will be provided by Daniel Arnström. This plug-in will work as a node in ROS. It might need some modification to be used in the project since it is not designed specifically for the LEGO truck. This modification will include implementing the functionality of sending control signals to the plug-in from the simulator. A map of Stora Visionen in Gazebo will be provided by the *Drone Project* group. This is another group that will take the same course, cooperation with this group will therefore be necessary for the development of the simulator. The functionality of the LEGO truck will be simulated on its own before merged with the map from the Drone Project.

5 Migration to Visionen

The setup for the system with car and semi-trailer truck will be the same when moved to Lilla Visionen since the same hardware will be installed and used in the new location. The date for this move will be set in collaboration with the customer, orderer and advisor. The physical move of the hardware will be done with help from the LiU technical support and service (TUS).

The LEGO truck will at first be used in the lab using the setup from previous years with markers and IR cameras and a PC for the positioning system. When moving to Stora Visionen the positioning system for Stora Visionen will be used instead. This positioning system will use the same technique as in the lab but with different markers, IR cameras and software. The software used will be the Qualisys Track Manager (QTM), found in [2]. This is preinstalled on the computers in Stora Visionen. The markers will be used to be able to track those points on the object in real time. The setup will require some calibration using a stick that is moved around in Stora Visionen and this procedure has to be done every day the system will be used.

The positioning system in Stora Visionen will result in a vector of determined positions on the object as well as estimated velocities in different directions, both raw data and filtered. This data will be sent in real-time via ROS and will be used in the Linux computer to find the position of the LEGO truck in the room. The setup in the lab as well as in Lilla Visionen will be configured to send the same data as in Stora Visionen. This means that our system will not need to be reconfigured between the work areas since the input from them will be similar.

6 LEGO truck

The LEGO truck is a 2-trailer system built with LEGO and a LEGO EV3 unit as its core. The plan is for the truck to be able to follow a pre-calculated trajectory in both forward and backward motion. The system will be able to automatically plan and execute advanced parking maneuvers, i.e. parallel parking and reversing up to a loading-bay.

6.1 LEGO truck overview

The LEGO truck system will consist of three parts: the truck itself, a Linux computer running the motion planner and the positioning system. Figure 5 presents a rough sketch of the LEGO truck system when used in Stora Visionen. The motion planner will be implemented outside the EV3-unit to reduce its workload.

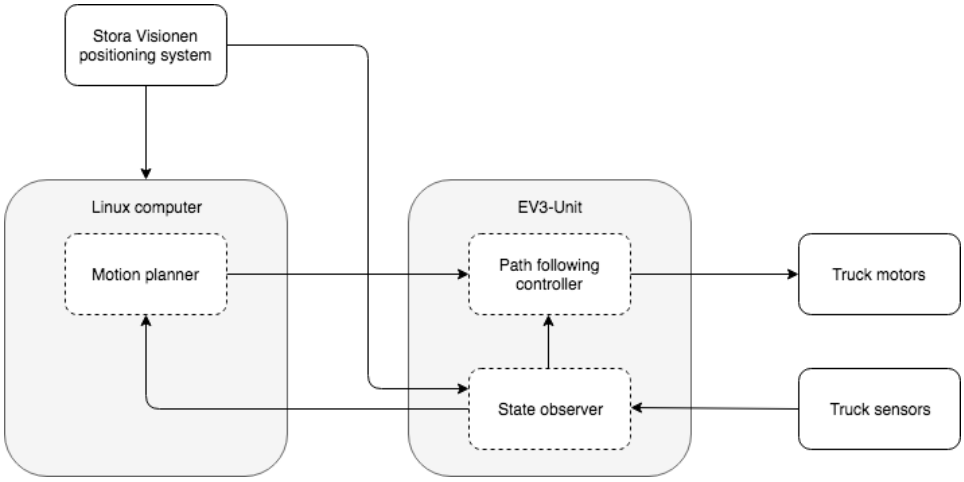


Figure 5: Detailed overview for the LEGO truck when in Stora Visionen.

When the LEGO truck will be used in the lab or in Lilla Visionen it will be used as described in figure 6. Both positioning systems will use IR cameras and markers on objects, but the difference will be in the type of markers and the IR camera setup. In either case the output from the positioning system will be the input to a ROS-node in the Linux program. This input will be the same in terms of position and velocities for either configuration only changing depending on what objects are on the field.

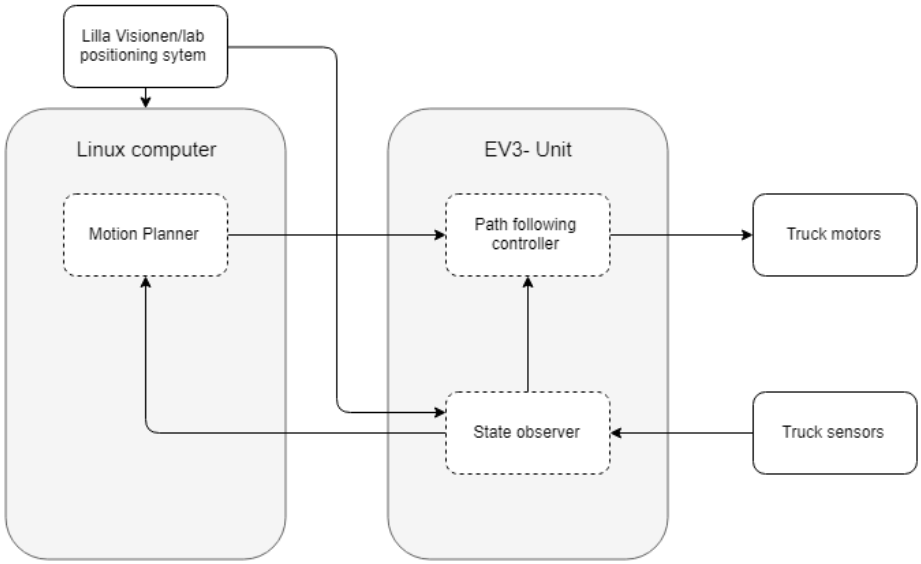


Figure 6: Detailed overview for the LEGO truck when in the lab or in Lilla Visionen.

6.2 Hardware

The LEGO truck will use the following hardware:

- EV3-unit with micro-SD card.
- Two LEGO servo motors.
- Two external LEGO angle sensors for measuring angles.

6.2.1 EV3

The LEGO truck will be controlled using a EV3-unit. The EV3 will run EV3dev which is a Linux-based operating system. A program will then run under this operating system and be connected to a Linux computer wirelessly. The program will also control the motors and take measurements from the sensors connected to it. Some more functionality might be transferred from the Linux computer to the EV3-unit depending on how it handles the workload.

6.2.2 Motors

The LEGO truck is equipped with two servo motors. One of the motors is used to steer the truck and the other motor is used for propulsion. Both motors are equipped with an angle sensor

6.2.3 Sensors

The LEGO truck is equipped with two angular sensors that are measuring the relative angles between the truck and the trailer. In figure 7, the angle between the truck and dolly is denoted as β_2 and the angle between the dolly and trailer is denoted as β_3. All sensors will measure angles relative to where they started, i.e they do not know their starting angle.

6.3 Motion model

A schematic description of the Lego truck is illustrated in figure 7. The Lego truck is a general 2-trailer system, as presented in [3].

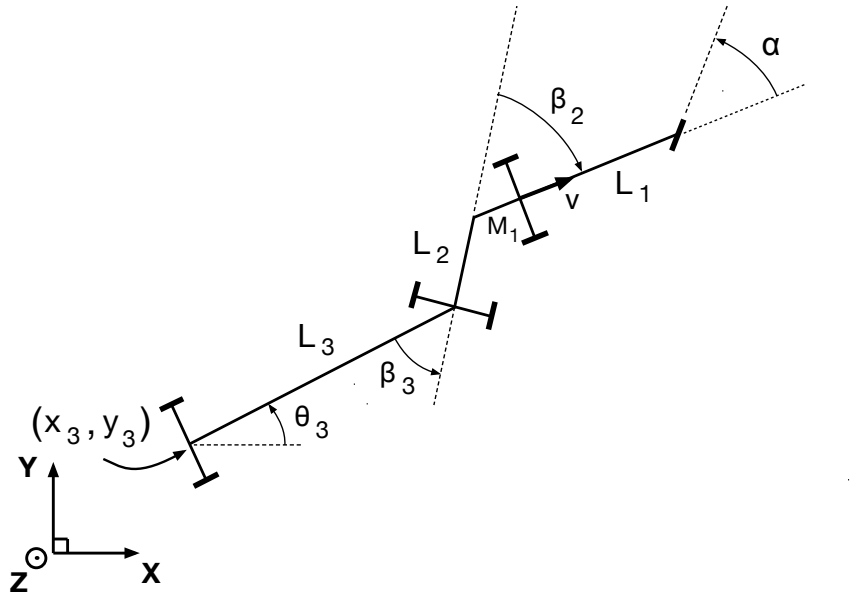


Figure 7: Schematic view of the LEGO truck, as presented in [3].

Here x_3 and y_3 is the position of the center of the trailer's rear axle, v is the velocity of the truck, θ_3 is the absolute angle of the trailer, β_3 is the angle between the trailer and the dolly, β_2 is the angle between the dolly and the truck and α is the steering angle (control signal) of the truck. L_3 is the distance between the axles of the trailer and the dolly, L_2 is the distance between the dolly-axle and the connection point between the dolly and the truck. L_1 is the distance between the axles of the truck and M_1 is the distance between the truck's rear-axle and the connection point to the dolly. v is the velocity of the rear axle of the truck. The system moves forward when $v > 0$ and backwards when $v < 0$. The system can then be represented by the generalized coordinates $\mathbf{p} = [\, x_3 \quad y_3 \quad \theta_3 \quad \beta_3 \quad \beta_2 \,]^T$. The 2-trailer truck can then be modeled as:

$$\dot{x}_3 = v\cos\beta_3\cos\beta_2\left(1 + \frac{M_1}{L_1}\tan\beta_2\tan\alpha\right)\cos\theta_3 \tag{1a}$$

$$\dot{y}_3 = v\cos\beta_3\cos\beta_2\left(1 + \frac{M_1}{L_1}\tan\beta_2\tan\alpha\right)\sin\theta_3 \tag{1b}$$

$$\dot{\theta}_3 = v\frac{\sin\beta_3\cos\beta_2}{L_3}\left(1 + \frac{M_1}{L_1}\tan\beta_2\tan\alpha\right) \tag{1c}$$

$$\dot{\beta}_3 = v\cos\beta_2\left(\frac{1}{L_2}\Big(\tan\beta_2 - \frac{M_1}{L_1}\tan\alpha\Big) - \frac{\sin\beta_3}{L_3}\Big(1 + \frac{M_1}{L_1}\tan\beta_2\tan\alpha\Big)\right) \tag{1d}$$

$$\dot{\beta}_2 = v\left(\frac{\tan\alpha}{L_1} - \frac{\sin\beta_2}{L_2} + \frac{M_1}{L_1L_2}\cos\beta_2\tan\alpha\right) \tag{1e}$$

The model is assumed to operate on flat surfaces and valid under a no-slip condition.

6.4 Motion planner

A lattice-based motion planner will be developed based of the solution presented in [3]. The motion planner will use a pre calculated set of motion primitives to create a reference trajectory that will be executed by the path following controller.

The idea behind lattice-based motion planning is to sample the state space of the vehicle and constrain the vehicle's motion to a so called *lattice graph* $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ which is a graph that is embedded in euclidean space $\mathbb{R}^n$ forming a regular tiling. Each node $v \in \mathcal{V}$ represents a discrete state of the vehicle and each edge $e \in \mathcal{E}$ encodes a motion that satisfies the constraints imposed on the LEGO truck.

6.4.1 State space discretization

The discrete states in the lattice graph $\mathcal{G}$ defines the states the LEGO truck can reach with the vehicle's constraints encoded in the set of motion primitives. Each valid state can then be represented with the state vector $\mathbf{s} = (x_3^d, y_3^d, \theta_3^d, \alpha^d)$ where x_3^d and y_3^d is constrained to a grid with a resolution of $r = 0.1m$, θ_3^d is discretized into a set of 16 unique angles spread around a circle and α^d is limited to one angle, $\alpha^d = 0$.

6.4.2 Motion primitives

The motion primitives are generated by solving the following optimal control problem, as described in [3],

$$
\begin{aligned}
\underset{u(\cdot),\,T}{\text{minimize}} \quad & \int_0^T f_0(\bar{\boldsymbol{p}}(t), u(t))dt \\
\text{subject to} \quad & \dot{\boldsymbol{p}}(t) = f(\boldsymbol{p}(t), \alpha(t)), \\
& \dot{\alpha}(t) = \omega, \;\; \dot{\omega}(t) = u, \\
& \bar{\boldsymbol{p}}(0) = (\boldsymbol{p}_i^d, \alpha_{e,i}^d, 0), \;\; \bar{\boldsymbol{p}}(T) = (\boldsymbol{p}_f^d, \alpha_{e,f}^d, 0), \\
& \mid \beta_3(t) \mid \leq \beta_{3,max}, \;\; \mid \beta_2(t) \mid \leq \beta_{2,max}, \\
& \mid \alpha(t) \mid \leq 0.8\alpha_{max}, \;\; \mid \omega(t) \mid \leq \omega_{max}, \\
& \mid u(t) \mid \leq u_{max}
\end{aligned}
\tag{2}
$$

where $\bar{\boldsymbol{p}}(t), u(t))$ is the objective function chosen as $\bar{\boldsymbol{p}}(t), u(t)) = 10\omega^2 + u^2$ which generates smooth steering angles when possible and $\dot{\boldsymbol{p}}(t) = f(\boldsymbol{p}(t), \alpha(t))$ is the vehicle model for the LEGO truck. The constraints $\bar{\boldsymbol{p}}(0) = (\boldsymbol{p}_i^d, \alpha_{e,i}^d, 0)$ and $\bar{\boldsymbol{p}}(T) = (\boldsymbol{p}_f^d, \alpha_{e,f}^d, 0)$ are the initial and final state. The steering angle rate at the initial and final state is zero to ensure that the steering angle is a C^1-function even when different primitives are combined. The constraints on the trailer angles, steering angle, steering angle rate and acceleration are $\beta_{3,max} = \pi/2$, $\beta_{2,max} = \pi/2$, $\alpha_{max} = \pi/4$, $\omega_{max} = 1.5$, $u_{max} = 40$. To make sure that no generated primitives saturates the steering angle the constraint for α is tightened by an additional 20%. The optimal control problem is solved using the CasADi toolkit, described in [5].

To make the generation of motion primitives more effective and decrease computation time all primitives are not generated. Only primitives to grid points in specif-

ically selected sets are generated both for forward and backward motion. This is possible due to the symmetry result explained in [3].

6.4.3 Lattice planner

The motion primitives generated offline will be used in the lattice planner to create a reference trajectory from a start state to a goal state. The problem for the lattice planner is defined by an initial state $\mathbf{s}_{start}$, goal state $\mathbf{s}_{goal}$ and a representation of the world, $\mathcal{W}$ which includes all known obstacles. A solution is viable if the start and goal states are connected by a collision-free sequence of motion primitives.

To find the optimal sequence between the start and goal state a search algorithm ($A*$ together with a suitable admissible heuristic to guarantee optimality) will be applied to find the path with minimal cost. This sequence is then sent to the path following controller.

6.4.4 Interface between lattice planner and path following controller

The interface between the motion planner and the path following controller for the LEGO truck will be in global coordinates.

$$[x_{3,0}, y_{3,0}, \theta_{3,0}, \beta_{3,0}, \beta_{2,0}, u_0, v_0] \tag{3}$$

where $x_{3,0}$ is the reference coordinates for the rear axle of the LEGO truck, $\theta_{3,0}$ is the reference angle of the trailer, $\beta_{3,0}$ is the reference angle between the trailer and dolly, $\beta_{2,0}$ is the reference angle between dolly and truck, $u_0 = \tan\alpha_0$ that is the reference control signal and v_0 is the reference velocity.

6.5 Path following controller

The path following controller will be implemented on the EV3-unit if possible. If it turns out not to be possible it will be implemented on the Linux computer instead. The path following controller will be used as presented in [4]. Based on a model of the LEGO truck the controller will stabilize the center of the rear axle of the trailer along the chosen trajectory during backward motion and the center of the rear axle of the truck for forward motion.

The system of equations (1) can be represented as $\dot{\mathbf{p}} = v_3 f(\mathbf{p}, \alpha)$, where v_3 is the velocity of the rear axle of the trailer. The conversion between v and v_3 is done using geometry and can be found in equation (4).

$$v_3 = v \cos\beta_3 \cos\beta_2 \Big(1 + \frac{M_1}{L_1} \tan\beta_2 \tan\alpha\Big) \tag{4}$$

It is possible to eliminate the velocity dependence from the model by using time scaling and we can therefore assume v_3 to be a predetermined constant. In order to avoid the singularities in system (1), the angles needs to be constrained to $|\beta_3| < \pi/2$, $|\beta_2| < \pi/2$ and $|\alpha| < \pi/2$. The steering angle α enters as $\tan\alpha$ everywhere in system (1) and the input substitution

$$u = \tan\alpha \tag{5}$$

is used to manipulate u instead of α. Because $\tan(\alpha) \in]-\infty, \infty[$ when $\alpha \in]-\pi/2, \pi/2[$ u does not have to be constrained to avoid the singularities in system (1). In practise however u needs to be constrained because the maximum steering angle, $|\alpha_{max}|$ is less than $\pi/2$. This is important to consider when the controller is designed.

The goal is to follow a nominal path which can be described as

$$\dot{\boldsymbol{p}}_0 = v_{3,0} f(\boldsymbol{p}_0(t), u_0(t)) \tag{6}$$

and the system in (1) can be described in terms of the deviation from the trajectory in equation (6). As in [4], a new parameter $s(t)$ representing the distance travelled by the trailers rear axle on the path up to a time t. Equation (6) can now be rewritten as:

$$\frac{d\boldsymbol{p}_0}{ds} = f(\boldsymbol{p}_0(s), u_0(s)) \tag{7}$$

and the curvature of the path followed by the trailer is

$$\kappa_0(s) = \frac{d\theta_{3,0}}{ds} = \frac{\tan\beta_{3,0}(s)}{L_3} \tag{8}$$

We introduce the error of the angles and control input relative to the path as:

$$\tilde{\theta}_3(t) = \theta_3(t) - \theta_{3,0}(s(t)) \tag{9a}$$
$$\tilde{\beta}_3(t) = \beta_3(t) - \beta_{3,0}(s(t)) \tag{9b}$$
$$\tilde{\beta}_2(t) = \beta_2(t) - \beta_{2,0}(s(t)) \tag{9c}$$
$$\tilde{u}(t) = u(t) - u_0(s(t)) \tag{9d}$$

and z_3 as the lateral deviation of the centre point of the trailer's rear axle and the reference trajectory.

Now the (1) can be represented in the Frenet frame as:

$$\dot{s} = v_3 \frac{\cos\tilde{\theta}_3}{1-\kappa_0(s)z_3} \tag{10a}$$

$$\dot{z}_3 = v_3 \sin\tilde{\theta}_3 \tag{10b}$$

$$\dot{\tilde{\theta}}_3 = v_3\left(\frac{\tan(\tilde{\beta}_3+\beta_{3,0})}{L_3} - \frac{\kappa_0(s)\cos\tilde{\theta}_3}{1-\kappa_0(s)z_3}\right) \tag{10c}$$

$$\dot{\tilde{\beta}}_3 = v_3\Bigg(\frac{\tan(\tilde{\beta}_2+\beta_{2,0}) - \frac{M_1}{L_1}(\tilde{u}+u_0)}{L_2\cos(\tilde{\beta}_3+\beta_{3,0})C_1(\tilde{\beta}_2+\beta_{2,0},\tilde{u}+u_0)} - \frac{\tan(\tilde{\beta}_3+\beta_{3,0})}{L_3} - \tag{10d}$$
$$- \frac{\cos\tilde{\theta}_3}{1-\kappa_0(s)z_3}\left(\frac{\tan\beta_{2,0}-\frac{M_1}{L_1}u_0}{L_2\cos\beta_{3,0}C_1(\beta_{2,0},u_0)} - \kappa_0(s)\right)\Bigg)$$

$$\dot{\tilde{\beta}}_3 = v_3\Bigg(\Bigg(\frac{\frac{\tilde{u}+u_0}{L_1\cos(\tilde{\beta}_2+\beta_{2,0})} - \frac{\tan(\tilde{\beta}_2+\beta_{2,0})}{L_2} + \frac{M_1}{L_1L_2(\tilde{u}+u_0)}}{\cos(\tilde{\beta}_3+\beta_{3,0})C_1(\tilde{\beta}_2+\beta_{2,0},\tilde{u}+u_0)}\Bigg) - \tag{10e}$$
$$- \frac{\cos\tilde{\theta}_3}{1-\kappa_0(s)z_3}\left(\frac{\frac{u_0}{L_1\cos\beta_{2,0}} - \frac{\tan\beta_{2,0}}{L_2} + \frac{M_1}{L_1L_2}u_0}{\cos\beta_{3,0}C_1(\beta_{2,0},u_0)}\right)\Bigg)$$

where

$$C_1(\beta_{2,0},u_0) = 1 + \frac{M_1}{L_1}\tan\beta_{2,0}u_0$$

The system (10), can be expressed with v, instead of v_3 by using equation (4). Equation (10a) is disregarded since there is no interest in controlling the speed of the LEGO truck. By introducing a new state vector $\boldsymbol{p}_e = [z_3 \quad \tilde{\theta}_3 \quad \beta_3 \quad \beta_2]$ with the coordinates for the path as $\boldsymbol{p}_{e,0} = [0 \quad 0 \quad \beta_{3,0} \quad \beta_{2,0}]$ and error state vector $\tilde{\boldsymbol{p}}_e = \boldsymbol{p}_e - \boldsymbol{p}_{e,0} = [z_3 \quad \tilde{\theta}_3 \quad \tilde{\beta}_3 \quad \tilde{\beta}_2]$, the nonlinear tracking error dynamics can then be expressed as

$$\dot{\tilde{\boldsymbol{p}}}_e = v f_e(\tilde{\boldsymbol{p}}_e, \tilde{u}, \boldsymbol{p}_{e,0}, u_0) \tag{11}$$

6.5.1 Linearization

The error model (10) is a nonlinear system and thus has to be linearized to apply LQ techniques. This can be achieved by linearizing around the stationary point $(\tilde{\boldsymbol{p}}_e \quad \tilde{u}) = (\bar{0} \quad 0)$ and using a reference path on the form of a straight line, $(\boldsymbol{p}_{e,0}, u_0) = (\bar{0}, 0)$, as shown in [4]. This yields the linear model

$$\dot{\tilde{\boldsymbol{p}}}_e = A\tilde{\boldsymbol{p}}_e + B\tilde{u} \tag{12}$$

with matrices A and B as

$$A = v\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{L_3} & 0 \\ 0 & 0 & -\frac{1}{L_3} & \frac{1}{L_2} \\ 0 & 0 & 0 & -\frac{1}{L_2} \end{bmatrix} \quad B = v\begin{bmatrix} 0 \\ 0 \\ -\frac{M_1}{L_1 L_2} \\ \frac{L_2+M_1}{L_1 L_2} \end{bmatrix} \tag{13}$$

The system is marginally stable in forward motion and, as expected unstable in backward motion. This is shown by the system's characteristic polynomial:

$$\det(\lambda I - A) = v^2\lambda^2\left(\lambda + \frac{v}{L_3}\right)\left(\lambda + \frac{v}{L_2}\right) \tag{14}$$

6.5.2 LQ-controller

The controller is based on previous years work where a hybrid controller with a LQ control-law is used to stabilize the system. The controller will change depending on which direction the LEGO truck is moving as can be seen in equation (15).

$$u = u_0 + \begin{cases} K_{\text{fwd}}\tilde{p}_e, & v > 0 \\ K_{\text{rev}}\tilde{p}_e, & v < 0 \end{cases} \tag{15}$$

The feedback gains K_{fwd} and K_{rev} are designed using LQ-techniques. Given the cost function,

$$J = \int_0^\infty (x^T Q x + \tilde{u}^2)dt \tag{16}$$

where Q is a positive semi-definite matrix that weights the states such that the requirements are met. The optimal feedback gain is given by:

$$K = -B^T P \tag{17}$$

where P is the solution to the Algebraic Riccati Equation:

$$A^T P + PA + Q = PBB^T P \tag{18}$$

6.5.3 State observer

The state observer will use the initial measurement from the external positioning system to get a starting point for the truck. This will provide the initial global position of the rear axle of the trailer (x_3, y_3) and the initial measurement for β_2, β_3 and θ_3. The steering angle α will not be able to be detected by the external positioning system. One way of handling this will be to run a script at start-up and moving the steering axle manually until it looks centered. Another way of doing it is to save the last measurement from the last run of the program in a file and loading from this at start-up. The solution found most reliable among these will be used.

The observer will use the internal measurements, the measurements from the LEGO sensors that measure angles, together with the measurements from the external positioning system. These will be combined with the motion model presented in equation (1) to estimate the states. If the measurements from the LEGO sensors provide accurate estimates, the external measurements could be used only at start up to initialize the measurements. If the estimates quickly start drifting from the true states when using only the LEGO sensors, the two measurement systems could be combined for more accurate estimates. The motion model together with the measurements is as follows:

$$\dot{x}_3 = v\cos\beta_3\cos\beta_2\left(1 + \frac{M_1}{L_1}\tan\beta_2\tan\alpha\right)\cos\theta_3 \tag{19a}$$

$$\dot{y}_3 = v\cos\beta_3\cos\beta_2\left(1 + \frac{M_1}{L_1}\tan\beta_2\tan\alpha\right)\sin\theta_3 \tag{19b}$$

$$\dot{\theta}_3 = v\frac{\sin\beta_3\cos\beta_2}{L_3}\left(1 + \frac{M_1}{L_1}\tan\beta_2\tan\alpha\right) \tag{19c}$$

$$\dot{\beta}_3 = v\cos\beta_2\left(\frac{1}{L_2}\left(\tan\beta_2 - \frac{M_1}{L_1}\tan\alpha\right) - \frac{\sin\beta_3}{L_3}\left(1 + \frac{M_1}{L_1}\tan\beta_2\tan\alpha\right)\right) \tag{19d}$$

$$\dot{\beta}_2 = v\left(\frac{\tan\alpha}{L_1} - \frac{\sin\beta_2}{L_2} + \frac{M_1}{L_1L_2}\cos\beta_2\tan\alpha\right) \tag{19e}$$

$$h_1 = \beta_3 \tag{19f}$$

$$h_2 = \beta_2 \tag{19g}$$

$$h_3 = \alpha \tag{19h}$$

$$h_4 = x_3 \tag{19i}$$

$$h_5 = y_3 \tag{19j}$$

$$h_6 = \theta_3 \tag{19k}$$

$$h_7 = \beta_3 \tag{19l}$$

$$h_8 = \beta_2 \tag{19m}$$

Where the measurement equations $h_1 - h_8$ are all in discrete time. Equations $h_1 - h_3$ are measurements from the internal measurement system. Equations $h_4 - h_8$ are measurements from the external measurement system. The internal measurement system will have offsets, these will be calculated by the external positioning system together with the chosen method for α, all at time 0. The velocity v will be either 1 or -1 depending on if the truck is moving forward or backward. If a certain velocity is desired, v_{des}, this will be handled using time scaling. If we set the following variables:

$$x = [x_3, y_3, \theta_3, \beta_3, \beta_2]^T \tag{20a}$$

$$y_{int} = [h_1, h_2, h_3]^T \tag{20b}$$

$$y_{ext} = [h_4, h_5, h_6, h_7, h_8]^T \tag{20c}$$

$$u = [\alpha, v_{des}]^T \tag{20d}$$

$$\tag{20e}$$

We can write the model on the following compact form:

$$\dot{x} = f(x, u) \tag{21a}$$
$$y_{int} = h_{int}(x, u) + w_{int} \tag{21b}$$
$$y_{ext} = h_{ext}(x, u) + w_{ext} \tag{21c}$$

where w_{int} and w_{ext} are the measurement noises for the internal and external measurements respectively. An EKF filter can then be used on this model to estimate the states. This will use one measurement update for the external system and one for the internal system. Trigonometry can be used to get the position of different parts of the truck using the state estimates. At first the EKF filter will be implemented on the EV3. If it doesn't have the capability needed it will be used on the Linux computer instead and the measurements and estimates will be sent between the units. A flowchart for how the state observer will work can be seen in figure 8.

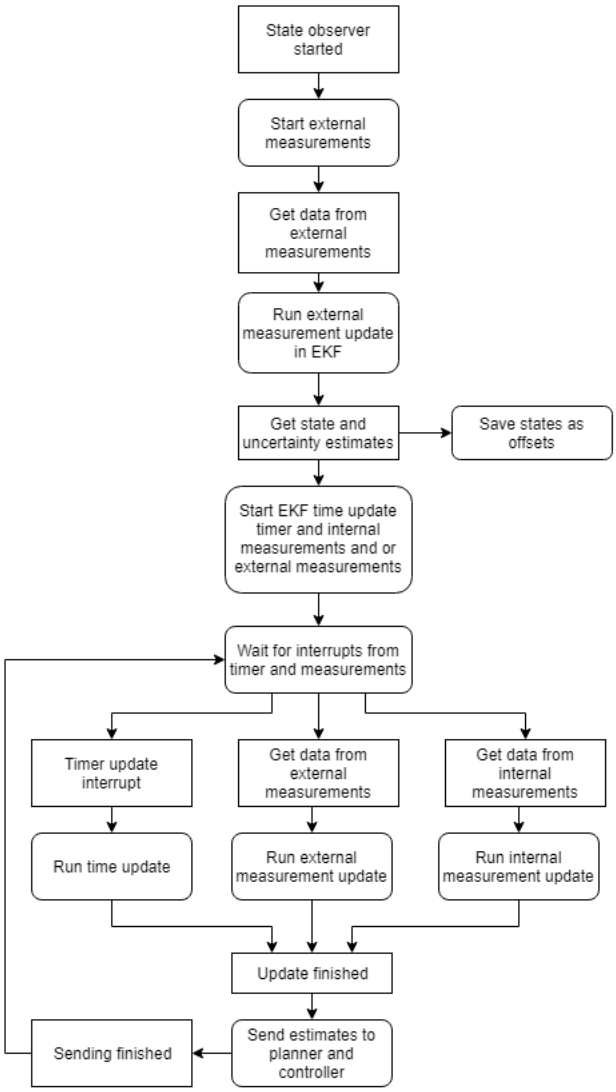


Figure 8: Flowchart for the state observer.

6.6 Advanced parking features

The LEGO truck will be able to execute advanced parking maneuvers such as parallel parking and reversing up to a loading bay. To do this the system will combine both forward and backward trajectories to move the LEGO truck to its goal position. The motion planner will be expanded in order to generate the set of trajectories needed for the different parking manoeuvres.

Two functions with associated buttons will be implemented. One called *Park by closest loading bay* and another called *Parallel park*. To test each of the parking functions outside the simulator a suitably scaled ”loading bay” and ”parking space” will be created using reflective tape and other materials. A figure of the loading bay situation can be seen in figure 9. The truck is supposed to reverse into a loading bay with the goal of having the back end of the rear trailer at the short end of the loading bay.

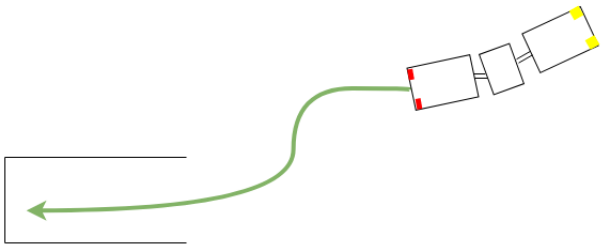


Figure 9: Sketch of the loading bay situation.

A figure of the parallel parking situation can be seen in figure 10. The goal of the parallel parking is to have the whole truck parked inside the parking space.

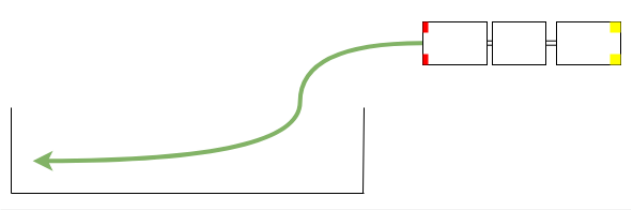


Figure 10: Sketch of the parallel parking situation.

A flowchart for how the loading bay situation will be can be seen in figure 11. The same overall flowchart will be used for the parallel park situation but with some different functionality for some of the segments of the flowchart.

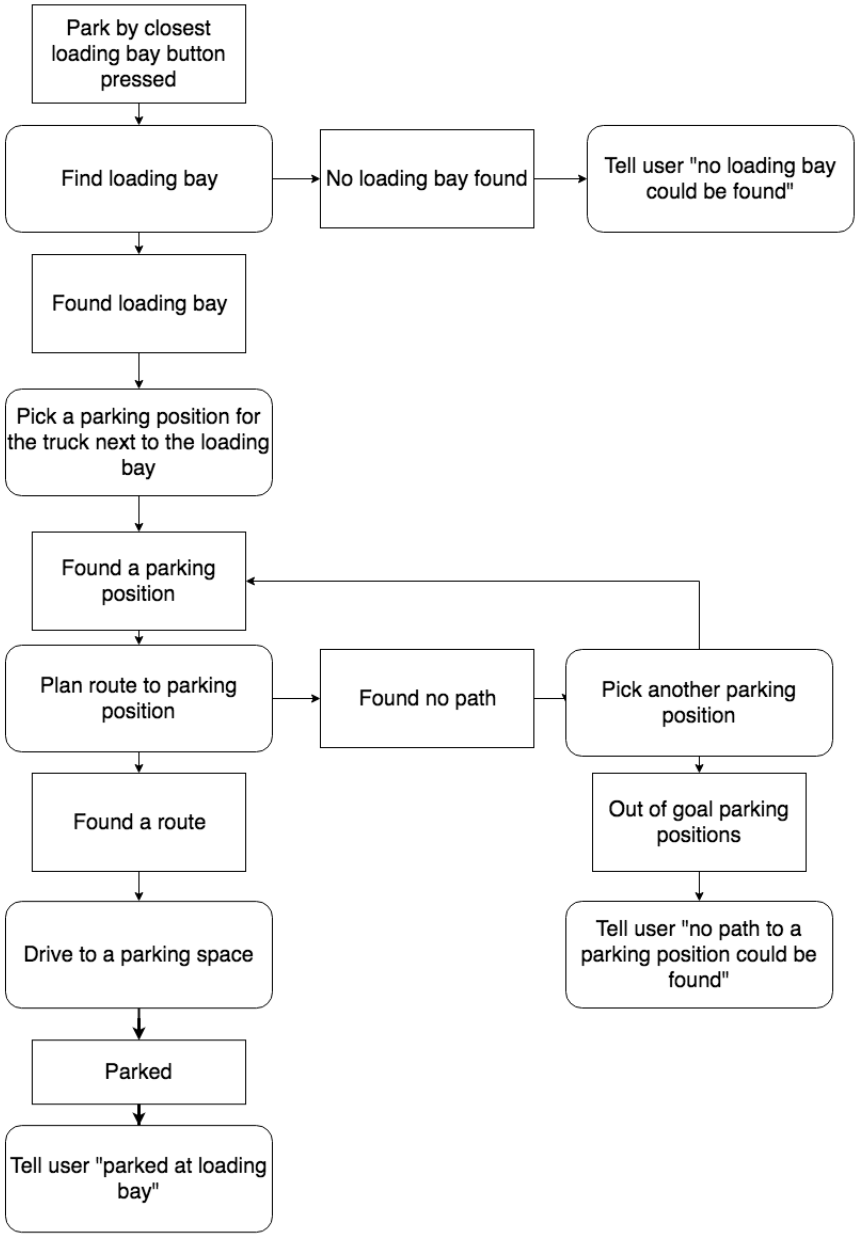


Figure 11: Flowchart of the loading bay situation.

6.7 Communication

The communication between the EV3 unit and the Linux computer will be done using bluetooth or wifi, whichever have more satisfying throughput and stability.

7 Parking feature for the car and semi-trailer truck

A new feature for the car this year will be the parking feature. This will be implemented for the car by expanding the motion planner from previous years. The same parking functionality will also be implemented for the semi-trailer truck if there is time for it.

When the motion planner cannot find a path forward due to a stationary object blocking the way or a moving vehicle suddenly stopping and blocking the path, the motion planner will try to back up to see if it can find another path. If the motion planner still cannot find a path, parking will be initiated. The program will then look for possible parking spaces and use the motion planner to find a suitable path to the parking space. The parking spaces should be close to the side of the road to get out of the way as much as possible. A flowchart for the parking feature when the road is blocked can be seen in figure 12. The plan for this feature is to have it automatically engage when necessary.

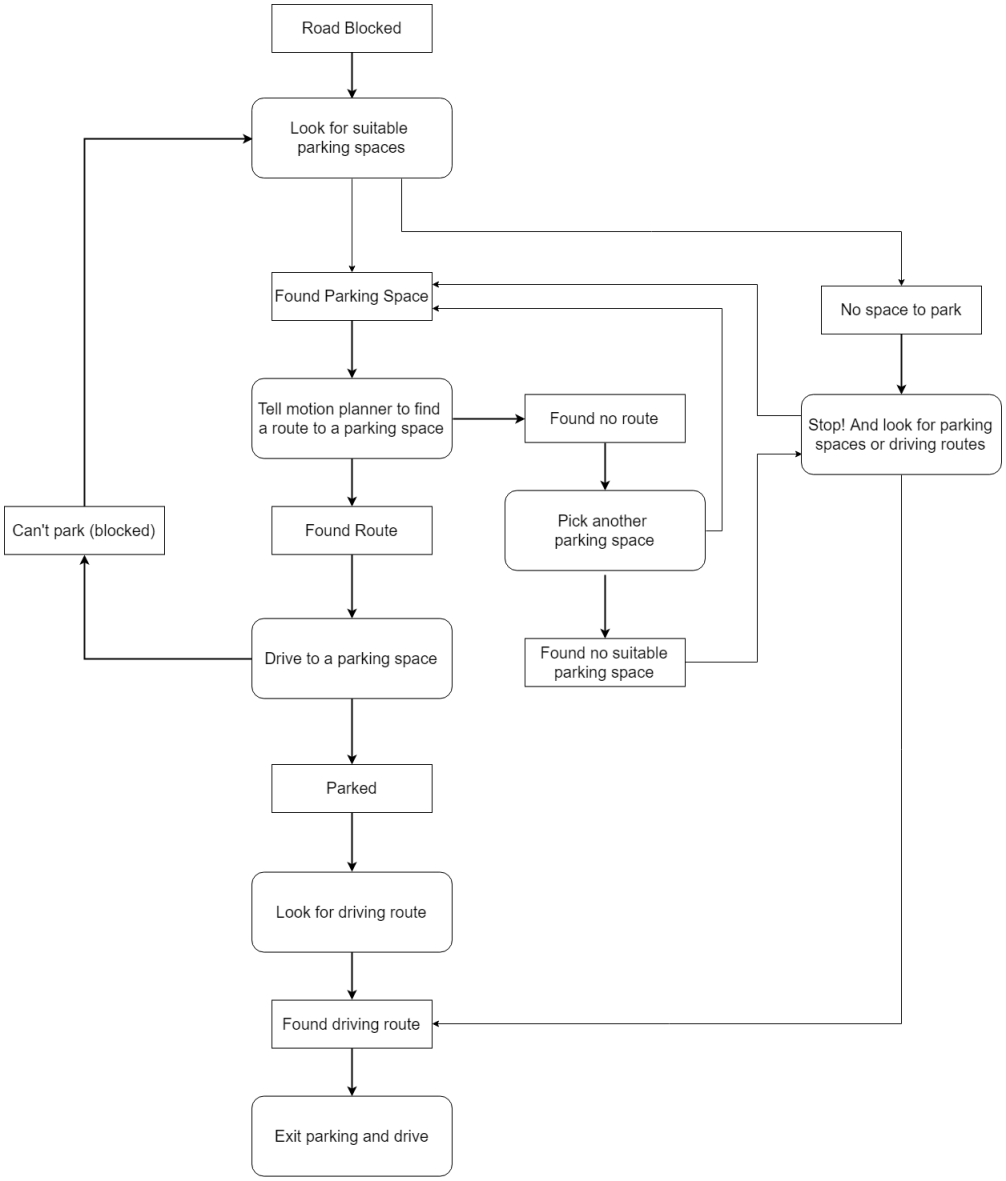


Figure 12: Flowchart for parking when the road is blocked.

The plan is to implement 2 buttons in the user interface: *park somewhere* and *park here* respectively. The button park somewhere button should tell the vehicle to park where the program finds it suitable. Park here is a command for the program to try and park where the user specifies it to park. The flowchart for these can be seen in figure 13 and figure 14. The main goal is to develop the feature called park when road is blocked. The other two features could be natural steps along the way and could be found useful for the user.

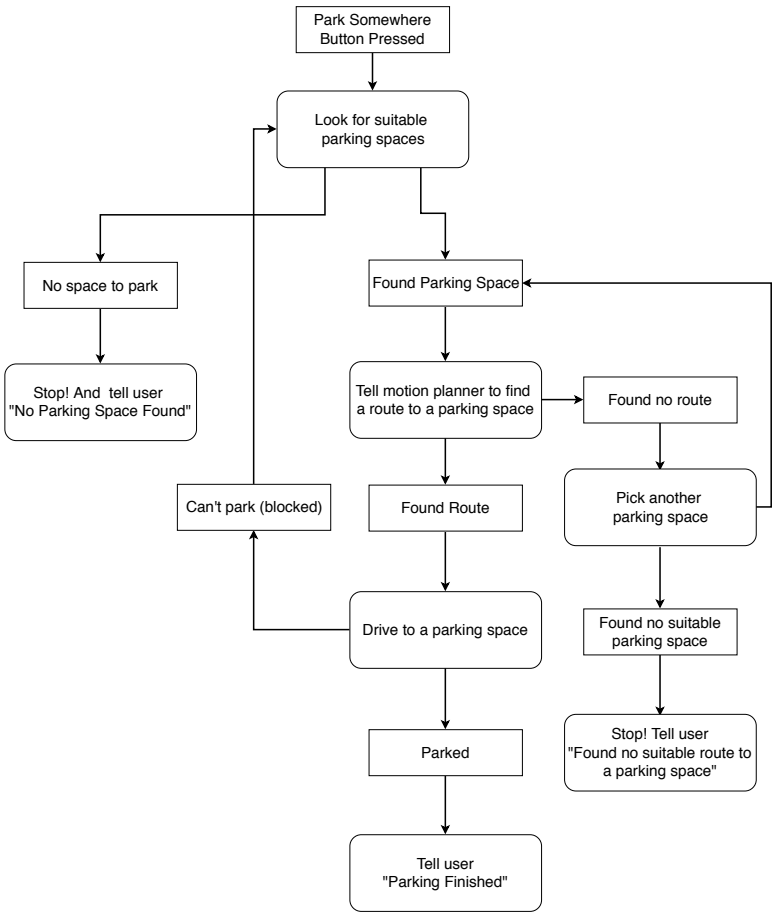


Figure 13: Flowchart for parking when park somewhere button is pressed

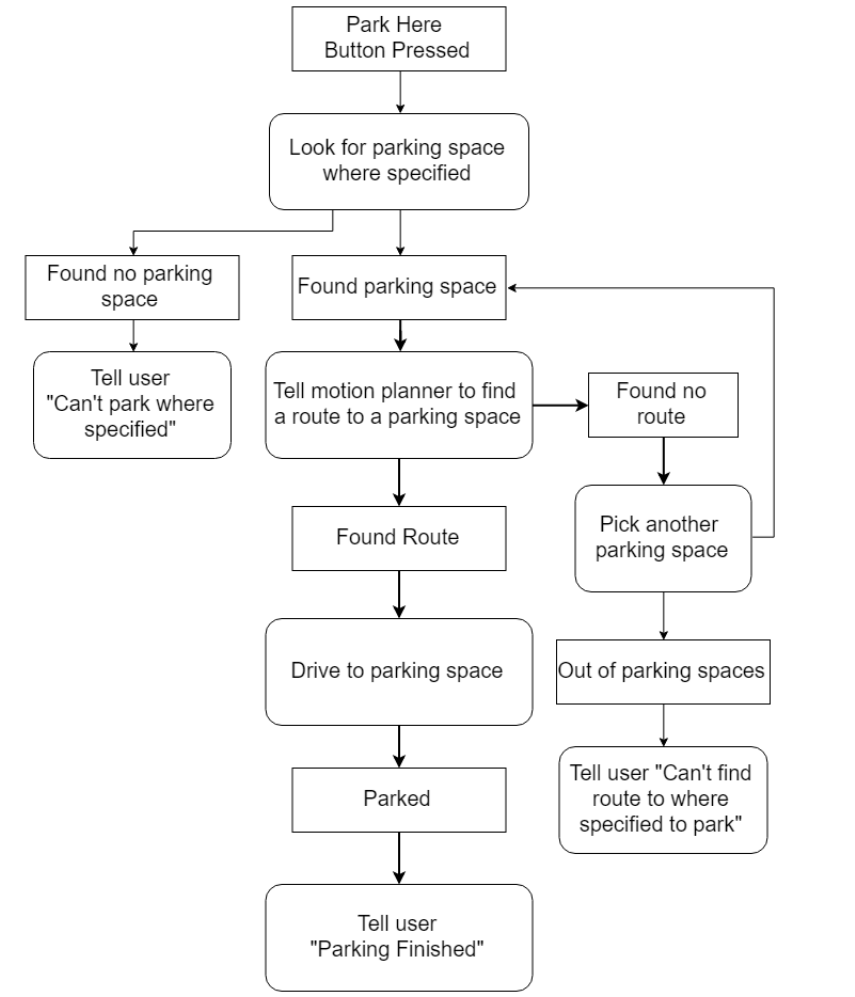


Figure 14: Flowchart for parking when the park here button is pressed

References

[1] D. Thomas, ”rqt”, Available at: http://wiki.ros.org/rqt

[2] Qualisys Track Manager, Available at: https://www.qualisys.com/software/qualisys-track-manager/

[3] O. Ljungqvist, N. Evestedt, M. Cirillo, D. Axehill, and O. Holmer. ”Lattice-based Motion Planner for a General 2-trailer system,” in *2017 IEEE Intelligent Vehicles Symposium(IV)*, pages 819-824, June 2017.

[4] O. Ljungqvist, D. Axehill, A. Helmersson. ”Path following control for a reversing general 2-trailer system”, in *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 2455-2461, December 2016.

[5] J Andersson, J Gillis, G Horn, J Rawlings and M Diehl ”CasADi – A software framework for nonlinear optimization and optimal control” in *Mathematical Programming Computation*, 2018

[6] O. Ljungqvist ”Motion Planning and Stabilization for a Reversing Truck and Trailer System”, Master’s Thesis, Linköping University, Automatic Control, 2015