Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Motion Planning for Autonomous Car Manipulator

Vadym Ostapovych

Supervisor: doc. Ing. Tomáš Krajník
January 2022

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Ostapovych Vadym** Personal ID number: **483578**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Motion Planning for Autonomous Car Manipulator

Bachelor's thesis title in Czech:

Plánování pohybu pro autonomní mobilní manipulátor vozidel

Guidelines:

The aim of this work is to design, implement and experimentally test methods for planning the movement of a mobile manipulator. The manipulator must be able load cars, move them to the intended location and unload unload them there. This work will focus on a motion planning in spatially-constrained areas of a parking lot. This method will be integrated into the Robotic Operating System (ROS) and experimentally verified on a real platform.
1) Familiarize yourself with the Lipraco Phoenix vehicle and its control system.
2) Familiarize yourself with the Robotic Operating System (ROS).
3) Prepare a simulation environment to test the developed methods.
4) Get acquainted with the motion planning methods used in mobile robotics.
5) Select suitable methods and perform their preliminary tests in the simulator.
6) Based on the results of preliminary experiments, select a suitable motion planning method and implement it for the Lipraco Phoenix platform.
7) Integrate the method into the navigation system of this platform and verify it experimentally.

Bibliography / sources:

[1] Yoshiaki Kuwata et al.: Motion planning for urban driving using RRT. In IROS 2008.
[2] Summers, T. : Distributionally Robust Sampling-Based Motion Planning Under Uncertainty. In IROS 2018.
[3] Costa M.M. et al.: A Survey on Path Planning Algorithms for Mobile Robots. In ICARCS 2019.

Name and workplace of bachelor's thesis supervisor:

doc. Ing. Tomáš Krajník, Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **26.07.2021** Deadline for bachelor thesis submission: **04.01.2022**

Assignment valid until: **30.09.2022**

doc. Ing. Tomáš Krajník, Ph.D. Supervisor's signature	prof. Ing. Tomáš Svoboda, Ph.D. Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I want to thank my supervisor Tomáš Krajník for fully supporting me and providing all the necessary information during the Bachelor's thesis. Also, I'd like to thank Skoda and CTU for the opportunity to join and contribute to the Phoenix project. Big thanks to the Phoenix project team for their research and implementation of the ROS driver for the Phoenix mobile robot. And I'm very grateful to CTU schoolmates for supporting in time of troubles.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses

Prague, January 3, 2022

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 3. ledna 2022

Abstract

Mobile manipulators can be applied in different fields for completing basic daily things, and correct execution of their movement is crucial. One of the ways to do this is to implement basic motion planning strategies to make the robot more flexible to the environment and interact with it. This work bases on the joint project of Skoda and CTU called the Phoenix robot and aims to find suitable ways for completing the basic maneuvers of the Phoenix car manipulator in the defined domain. During the thesis, the first task was for a robot to follow the preset path. For this, the different path controllers were designed, implemented and experimentally evaluated. The next crucial task of this work was to find suitable path planners for navigating the robot in the narrow spaces of Skoda parking. The best ones were used to create the maneuver library for further application in the project which allow robot move autonomously in the parking space.

Keywords: Motion planning, mobile manipulator, path controlling, RRT, Dubin's curves, ROS, Gazebo simulation

Supervisor: doc. Ing. Tomáš Krajník
Center of Artificial Intelligence, FEL

Abstrakt

Mobilní manipulátory najdou uplatnění v různých každodenních činnostech. Jedním ze způsobů, jak dosáhnout jejich správné činnosti, je implementovat základní strategie pro plánování pohybu, neboť ty robotům umožňují přizpůsobit svůj pohyb jejich prostředí a interagovat s jejich okolím. Tato práce vychází ze společného projektu Škoda Auto a ČVUT nazvaného robot Phoenix a klade si za cíl nalézt vhodné metody pro vykonávání základních manévrů transportéru autonomních vozidel Phoenix. Během práce bylo prvním úkolem, aby robot sledoval předem danou dráhu. Za tímto účelem byly navrženy, implementovány a experimentálně vyhodnoceny různé metody řízení pohybu po zadané drázy. Dalším zásadním úkolem této práce bylo najít vhodné plánovače cest pro navigaci robota v úzkých prostorách parkovišť. Ty nejlepší metody byly integrovány do knihovny manévrů pro další použití v daném projektu.

Klíčová slova: Plánování pohybu, mobilní manipulátor, ovládání cesty, RRT, Dubinsovy křivky, ROS, Gazebo simulace

Překlad názvu: Plánování pohybu pro autonomní mobilní manipulátor vozidel

Contents

Figures

Tables

Chapter 1

Introduction

This chapter briefly introduces the thesis's motivation and defines the goals, giving an overview of modern mobile robotic problems.

1.1 Motivation and goals

One of the crucial tasks of modern robotics is finding effective interaction methods between human beings and the environment with the injection of automated systems. Robots systems are appliable in many fields, from industry and education to military and space. The field of research in this thesis is mobile robotics in the industry and logistics.
The main feature of mobile robots is moving around a specific environment without fixing it to one particular location. They work remotely and have difficulties with unexpected updates of the domain. That is why they require a precise and predefined environment to avoid wrong behavior. One of the subclasses of mobile robots is the `Automated guided vehicle` [1] which uses the navigation system based on the information parsed from sensors like vision cameras, lidar scanners, GPS to perform movement along wires and lines in 2D space. The navigation system aims to localize the robot and find the sequence of motions that lead the robot to the target position using planning methods. Motion planning can be divided into two types depending on the current activity of the robot: offline and online. The first one aims to find the path with the confidence that the further information of the environment will be static; briefly, the robot knows the whole environment and will find the path to the target if such exists, but the main disadvantage is unexpected changes of the environments which can lead to the misbehavior. The online way of planning is to find the target during robot moves and adjust all the negative impacts on the environment, but this requires a tremendous computational load on the system and a complicated architecture of the decision process. This thesis uses an approach of discretizing the path as the sequence of basic maneuvers calculated offline and choosing the best one in the path decision process during the online part. Such an approach will minimize the computation cost and better react to an unpredicted environment. In this thesis, the main aim is to apply the methods of path following and planning in the collision-free environment and implement the library of maneuvers

for further use of AGV Phoenix. The proposed maneuvers will explore the current navigation and localization system, which the Phoenix team tests on the platform.
The thesis first briefly introduces navigation problems and presents the theoretical background of implemented methods based on the related works. The next chapters are devoted to the implementation tools and experiments, discussing and comparing the achieved results.

Chapter 2

State of art

This chapter introduces the navigation overview in mobile robotics, related works for path controlling and motion planning, presents the created software tools based on these researches.

2.1 Navigation overview

Navigation [2] is the field of science aiming to reach the given position in the frame of reference. This thesis is devoted to the navigation of the ground robot, which localization parameters are orientation and position. The well-built navigation architecture has to always complete the following tasks

- self-localization.
- planning the path to the goal location.
- map-creating of the environment.

Self-localization is the ability to find the current `robot's configuration` which can be defined as the state of degrees of freedom DOF. The mobile planar robot is represented with position and orientation—the set of possible robot's configuration form configuration space or `C-space`. For odometry, planar robot `C-space` is three-dimensional, exploring the translational and rotational ability of the robot [3]. Path planning is the extension of navigation, which studies finding the path to the goal configuration. It requires the representation of the environment, which can be done with a map. In this thesis, the metric map is used for collision-free path planning. The most common approach is detecting the obstacles as the polygons in the topological or grid-based maps. Non-map navigation uses require complicated approaches, usually requiring visual sensors like the camera for parsing the information about the current location [4].

2.2 Related works

The navigation of AGV vehicles has been crucial in engineering since the 1950s when the first AGV was brought to the market by Barrett Electronics

of Northbrook. During the 20th century, the engineers were primarily looking to apply such vehicles in small, closed spaces that did not require complicated localization methods. With the development of lasers and satellite navigation, the growth of computer science fields like machine learning and computer vision, the area of AGV applications spread exponentially. One of the new fields of the 21st century in AGV applications is automated dense parking, which is caused by the increase of cars in urban areas. As an example, in 2015, german company Lodige Industries developed and implemented the most extensive automated system in Denmark with 1000 cars.
The group of researchers from Stanley Robotics led by [5] Philip Polack presented a new approach to dense automated parking which is applicable on the parking with a large number of parking lots and requires precise navigation, path following, and planning in the narrow lines. The members of the Phoenix team aim to transfer the experience of Polack's team for navigation on the Skoda parking. The basic concept and reference results are taken from the above paper. As mentioned, the whole task of this thesis may be divided into two separate parts: proposing the most appropriate controller for the path following and creating the motion planning strategy with further exporting the founded maneuvers to the library or database. In this section, a brief overview of implemented software for solving the described above tasks is presented.

2.3 Path controlling overview

The path following is the first task for robotic engineers before applying any proposed motion planning strategies. In this thesis, the four path controllers were proposed and implemented in simulation: Basic Point to Point, Pure Pursuit, Pose Controller, Model Predictive Controller(MPC). Point to Point was also tested on the real robot 5.4.
All path control strategies can be divided into explicit and implicit. The first one does not require the analysis of the dynamic model of the robot with the known control theory methods; they aim to find interrelation between the robot inputs and reference trajectory position. This approach is presented in the article of data scientist engineer Ding[6] where she proposes three types of path controller: Pure Pursuit, Stanley, and MDP. The work of Coulter [7] describes the geometry, tuning, and implementation of Pure Pursuit. Model predictive controller conception, theoretical background, and implementation for the following reference are presented in the article of Hungarian researchers led by Reda Ahmad. Point to Point steering controller for the nonholonomic robot is presented in the chapter of robotics guide by Tim Bower [9] from Kansas State polytechnic.
The implicit methods use the control theory methods to adjust the robot dynamic to the required behavior, in our case path following. These methods require the state-space model of the robot, but they are frequently used for different controlling tasks, which also leads to deep analysis of the system, which can be challenging to implement in the case of complicated state-space

models. For example, Klanchar [10] used the linear state-space for the close loop to enable the robot to follow the reference path and feedforward loop for reference inputs. Also, Vieira [11] presented both ways of controller strategies for a nonholonomic mobile robot, simple Point to Point Controller, Pose controller, and PID controller for following the reference path, but he didn't aim to follow the input reference. One of the most significant research on the topic of path following for mobile robotics is presented in the book of Corke [12] who is the author of most modern robotics toolboxes for MATLAB. In the chapter on Mobile robot vehicles, he presented the wheeled mobile robots and control theory for them, which was implemented in Automated Driving Toolbox: Pose Controller, Point to Point, and Trajectory following with Stanley Controller. Some of the works of the presented authors were taken into account during the proposing of the controller in this thesis.

2.4 Motion planning research

Motion planning requires the finding of valid sequences of robot configuration from init configuration to the final one. Many authors have done great research on this topic in the past 60 years. Generalized information about the methods for solving motion planning problems can be founded in the book of Latombe [13] where he summarizes the term of configuration space, describes the methods of the 20th century for solving motion planning with most commonly used algorithms which based on grid-based search, potential artificial fields, and sampling-based algorithms. The more modern approach is described in the book Modern Robotics of Kevin M. Lynch [14], which also pays attention to an important part of trajectory generation and time optimality, metrics problems, summarizes all the experience of robotics engineers.
The motion planning for nonholonomic robots with constraints is described in the work of Pin[15], where he aims to find the method of computationally fast trajectories which will be deterministic and take into account the ability of forward and reverse motion of the vehicle in the environment with obstacles. The new approach to motion planning strategy also presented Lamiraux [16] where he considers the robot as a 4-D kinematic system for generating the path and the sane 3-D system for finding collision-free paths. Many of the described motion planning methods in the work of these authors were also implemented in navigation software.

2.5 Created software tools based on the related works

The research in the area of path following and planning found practical application in some of MATLAB toolboxes like Navigation [17], and Automated Driving [18], where the user can tune and adjust path controllers for the required robot kinematic model on the grid-based maps or plan the path with presented planners. In ROS, the motion planning package for the arm

robot is also presented and now is being developed and modified. Robotics engineer Sakai implemented sample robotics algorithms in Python on his Github page[19], which gives the programmer the practical understanding of using these algorithms. Another Python-based module is efficient in the context of control theory implementation in Python Control Systems Library[20], and anyone can find the vehicle steering dynamic control example on their pages. Phoenix team led by Tomáš Krajník presented their approach to solving motion planning problems from implementing simple grid-based search algorithms and ending with a navigational system called BearNav. All the code for the Phoenix robot is implemented in ROS driver written in C++. Summarizing, the most illustrative solutions for path following and planning problems can be observed in MATLAB toolboxes, but they can be used on the actual platform, as in our case is ROS, only after integration to MATLAB ROS toolbox, which is not practically used due to computational load and problems with code transferring. The work of Sakai is illustrative and practically understandable but requires the extra implementation in ROS, the same as Python Control Systems Library. Most of the code for mobile robotics is written in C or C++ languages for a faster communication, but its sources are usually limited.

Chapter 3

Theoretical concepts

This chapter introduces the theoretical background for the proposed path following and planning algorithms, describes the mathematical model of the robot.

3.1 Kinematic description of the robot

The Phoenix 5.15 is represented with a car-like model based on the use of Ackermann steering geometry with the center point in the axle of the front wheels. It is an adaptation of the non-holonomic bicycle model.

3.1.1 Kinematic equations

The robot kinematics is described using the following differential equations

$$\dot{\boldsymbol{x}} = v\,cos(\theta + \psi) \tag{3.1}$$
$$\dot{\boldsymbol{y}} = v\,sin(\theta + \psi) \tag{3.2}$$
$$\dot{\boldsymbol{\theta}} = \frac{v}{L} sin(\psi) \tag{3.3}$$
$$\dot{\boldsymbol{v}} = a \tag{3.4}$$
$$\dot{\boldsymbol{\psi}} = \sigma \tag{3.5}$$

where

1) Outputs of the systems, the pose of the robot in the 2D, on the plane
   - (x, y) - position of the center point.
   - $\theta \in [-\pi, \pi)$ - orientation relatively to the world frame.
2) Inputs to the system, robot's
   - $v \in [-3; 3]\ \frac{m}{s}$ - forward speed of the front wheels.
   - $\psi \in [-\pi/2; \pi/2]$ - steering angle of the front wheels, defines the robot rotational radius 3.2.
3) Parameters calculated from desired robot's inputs from the equation (3.8)

- $a \in [-a_{max}; a_{max}]\ \frac{m}{s^2}$ - rolling acceleration.
- $\sigma \in [-\sigma_{max}; \sigma_{max}]\ rad/s$ - steering velocity.

3.1.2 Constraints and parameters

From the system equations, the non-holonomic constraint can be withdrawn

$$\frac{\dot{x}}{cos(\theta + \psi)} = \frac{\dot{y}}{sin(\theta + \psi)} = v \tag{3.6}$$

also can be written in Pfaffian form [1]

$$\dot{x}\, sin(\theta + \psi) - \dot{y}\, cos(\theta + \psi) = 0 \tag{3.7}$$

It represents the fact that the robot cannot move perpendicularly to the forward speed vector.The desired inputs given on the system are

$$\vec{\boldsymbol{u}} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{v_{set}} \\ \boldsymbol{\psi_{set}} \end{bmatrix} \tag{3.8}$$

The inputs given on the robot wheels depend on the previous inputs. In every step of the simulation, the discrete derivation of inputs is calculated

$$\dot{u}_i(t_1) = \frac{\Delta u_i}{\Delta t} = \frac{u_i(t_1) - u_i(t_0)}{t_1 - t_0} \tag{3.9}$$

which represents the rolling acceleration a and angular velocity of steering σ. The sign of these parameters depends on the newly received value of the input

$$v = sgn(\Delta v)\, \min(\frac{|\Delta v|}{\Delta t}, a_{max}) \tag{3.10}$$

$$\sigma = sgn(\Delta \psi)\, \min(\frac{|\Delta \psi|}{\Delta t}, \sigma_{max}) \tag{3.11}$$

For the limiting the inputs changing rate the speed ramp is used, its parameters were identified experimentally

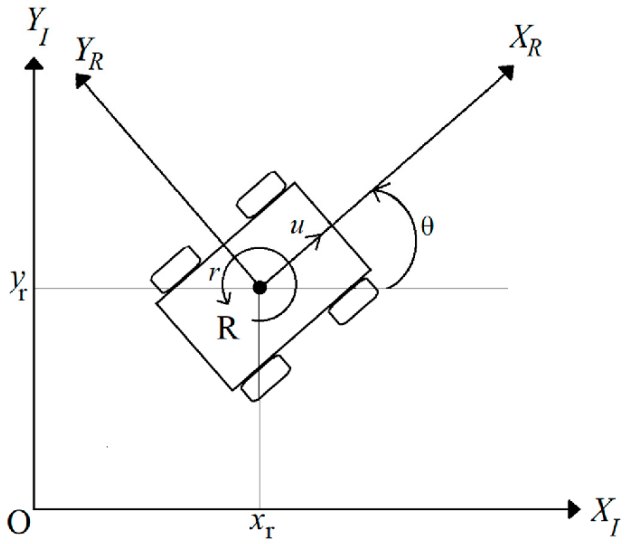
Changing rate name	Symbol	Value
Maximal rolling acceleration	a_{max}	$0.7[m/s^2]$
Maximal steering velocity	σ_{max}	$0.7[rad/s]$

Table 3.1: Maximal inputs changing rates

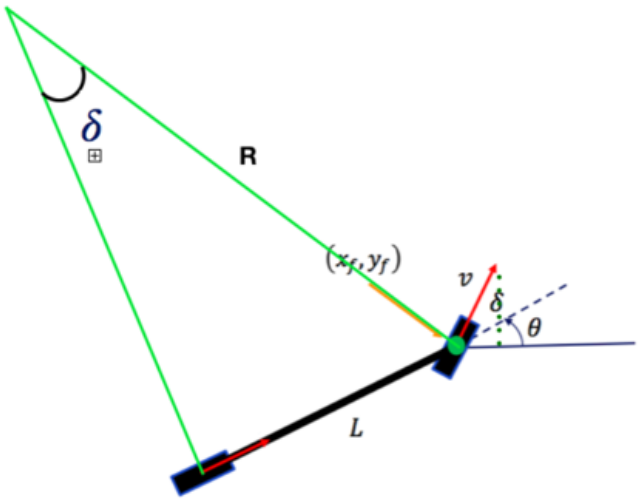
3.1.3 Transformation between unicycle model and car-like model

The 2D unicycle robot 3.1a represents the simplified car-like model. It based on the following system of equations

[1] $\sum_{i=1}^{N} a_i\, dq_i + b\, dt = 0$, where q_i - coordinates, a_i, b - constants



(a) : Unicycle model



(b) : Car-like model of the Phoenix robot

Figure 3.1: Comparing the unicycle and car-like models

$$\dot{x} = v\,cos(\theta) \tag{3.12}$$

$$\dot{y} = v\,sin(\theta) \tag{3.13}$$

$$\dot{\theta} = \omega \tag{3.14}$$

For applying the Ackermann steering geometry [23] the reference point is chosen somewhere on the wheelbase, in our case in the axle of the front wheels and defining the new robot state in the form

$$s(t) = [x(t), y(t), \theta(t), \psi(t), v(t)] \tag{3.15}$$

Based on the kinematics equations of the car-like model 3.1.1, our target is

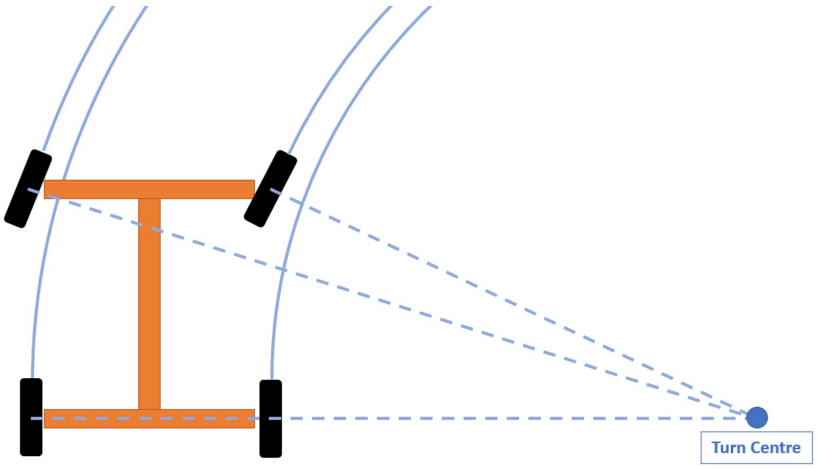


Figure 3.2: The steering geometry and the turn radius of the vehicle

to find the transformation between the unicycle model and the car-like model. We will apply Instantaneous Center of Rotation(ICR) illustrated on the figure 3.2 for the front wheels, which radius vector should be perpendicular to the forward speed v.

Robot's orientational angular velocity w depends on this radius in the vector and scalar forms.

$$\vec{v} = \vec{w} \times \vec{R} \implies w = \frac{v}{R} \tag{3.16}$$

Turn radius is calculated using the Ackermann geometry 3.1b like

$$R = \frac{L}{sin(\psi)} \tag{3.17}$$

By substituting the radius to the equation we obtain the transformation between orientational angular velocity of unicycle and to the steering angle for the car-like model

$$\dot{x} = v\, cos(\theta + \psi) \tag{3.18}$$

$$\dot{y} = v\, sin(\theta + \psi) \tag{3.19}$$

$$\dot{\theta} = \omega = \frac{v}{R} = \frac{v}{L} sin(\psi) \implies \psi = asin(\frac{\omega\, L}{v}) \tag{3.20}$$

The following constraints pay for the transformation (3.20)

$$-1 <= \frac{\omega\, L}{v} <= 1 \implies -\frac{v}{L} <= \omega <= \frac{v}{L} \tag{3.21}$$

The transformation between the coordinates speeds in the unicycle and car-like model is determined applying the trigonometric identities and written

with the 2D rotation matrix $\mathbf{R}(\alpha)$ in the form

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \mathbf{R}(\psi) \begin{bmatrix} v\,cos(\theta) \\ v\,sin(\theta) \end{bmatrix} \tag{3.22}$$

$$\psi = asin(\frac{\omega\, L}{v}) \tag{3.23}$$

$$\dot{\theta} = \omega \tag{3.24}$$

This approach is useful for the further path-following problems solutions where our target is to find the controller which will be able to follow the reference for the unicycle and then transform to a car-like model used on the real robot.

3.2 Path-following problem

In this subsection, the controller's theoretical background for the path following is described. The continuous trajectory for is defined as the sequence of the robot states $\boldsymbol{s_{ref}(t)}$ (3.15). For the controller proposing, we will assume that the target reference state defined as

$$\boldsymbol{s_{ref}(t)} = [x_{ref}(t), y_{ref}(t), \theta_{ref}(t), 0, v_{ref}(t)] \tag{3.25}$$

supposing that the reference of the steering angle is not followed, it is used only to follow the robot's other state's values. If the trajectory is represented, as the sequence of robot's poses, than the steering angle error can be reduced by close distances between generated poses (5.5).

3.2.1 Point to Point Controller

Point to Point Controller aims to find the control laws for the inputs that will allow the robot to follow the reference position $\boldsymbol{x} = [x_{ref}, y_{ref}]$ and front wheels rolling speed v_{ref}. Suppose the trajectory is the sequence of points. In that case, the distance between the two points in the sequence is the line. The orientation error can also be reduced by the dense generation of these points. The position error in the world frame is defined as

$$\boldsymbol{e_{world}} = \begin{bmatrix} x_{ref} - x \\ y_{ref} - y \end{bmatrix} = \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{3.26}$$

For the controller proposing, we'll go over the position errors in the local frame of the robot. With the linear transformation, we receive the longitudinal and lateral errors.

$$\boldsymbol{e_{robot}} = \begin{bmatrix} e_x \\ e_y \end{bmatrix} = \mathbf{R}(-\theta) \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{3.27}$$

where $\mathbf{R}(-\theta)$ - a counterclockwise 2D rotation matrix through an angle $-\theta$.

Steering control law

The idea of the steering law is to turn the robot precisely on the angle defined by errors from the equation (3.27). We will assume that this angle is defined in the same range as the steering angle, $[-\pi/2; \pi/2]$. The resulting control law is

$$\psi = atan2(e_y, |e_x|) \tag{3.28}$$

In case if the lateral error is zero, than the robot will be heading straight to the point.

Speed control law

For the speed control law, we will assume that the robot should reduce the distance error to the reference point

$$d = \sqrt{e_x^2 + e_y^2} = \sqrt{(\Delta x)^2 + (\Delta y)^2} \tag{3.29}$$

that does not depend on the error measured frame, as the rotation is a linear isometric transformation. The robot should approach the desired point with the speed of the same sign as the reference one and reduce it gradually with a reduction of the distance. We propose the proportional controller in the form.

$$v = v_{ref} + sgn(v_{ref}) K_v d, K_v > 0 \tag{3.30}$$

The problem of the presented controller is that in the case of the zero reference speed, the robot will stay in the same place. This can be solved by proposing the new sgn function depending on the reference speed.

$$\widetilde{sgn(v_{ref})} = \begin{cases} 1, \ v_{ref} >= 0 \\ -1, \ v_{ref} < 0 \end{cases} \tag{3.31}$$

which implies that if the reference speed is zero, let the robot approach the point with the positive speed. The controlling inputs should be saturated to the defined ranges. The result control law for both inputs is following

$$v = v_{ref} + \widetilde{sgn(v_{ref})) K_v \sqrt{e_x^2 + e_y^2}} \tag{3.32}$$

$$\psi = atan2(e_y, |e_x|) \tag{3.33}$$

Proposing of proportional constant K_v

The proportional constant K_v can be set experimentally. Here, we will derive its value from the braking distance of the vehicle. From the speed control (3.32) law

$$v - v_{ref} = \widetilde{sgn(v_{ref})) K_p d} \implies |\Delta v| = K_p d \implies K_p = \frac{|\Delta v|}{d} \tag{3.34}$$

Let us suppose that this member of the equation represents the speed change rate which should be zero when the robot approaches the target point. If

we assume that the maximum acceleration is 0.7 m/s^2, then we can use the uniformly accelerated motion distance.

$$x(t) = v_0 t - \frac{at^2}{2} = \frac{v(t^2) - v(t_0^2)}{2a} \tag{3.35}$$

For the target zero reference speed

$$x(t) = \frac{v(t_0^2)}{2a} \tag{3.36}$$

The maximal possible speed changing rate is $|\Delta v_{max}| = 3[m/s]$ due to the speed constraints 3.1.1, in case the robot moves straight to the reference point and needs to decrease the current speed from 3 to 0. Substituting to the equation (3.36)

$$d_{max} = \frac{(3\,m/s)^2}{2 \cdot 0.7\,m/s^2} = 6.4[m] \tag{3.37}$$

And here we can calculate the maximal possible value of proportional constant

$$K_v = \frac{|\Delta v_{max}|}{d_{max}} = \frac{3\,m/s}{6.4m} = 0.47 \tag{3.38}$$

When the speed change rate is less than the maximal robot acceleration a_{max}, the front speed will be decreasing smoother without final overshoots.

3.2.2 Pure Pursuit Controller

The idea of the pure pursuit controller is to reduce the lateral error of the robot by proposing the turn angle based on the look-ahead distance l_d from the axle of the front wheels. The radius R of the circle the robot should turn to follow the path can be determined using the geometry from the sines law based on the figure 3.3.

$$\frac{l_d}{sin(2\,\alpha)} = \frac{R}{sin(\frac{\pi}{2} - \alpha)} \tag{3.39}$$

$$R = \frac{l_d}{2\,sin(\alpha)} \tag{3.40}$$

By substituting the radius of the car-like model from the equation (3.17) the control law is given as

$$sin(\psi) = \frac{L}{R} = \frac{2\,L\,sin(\alpha)}{l_d} \implies \psi = asin(\frac{2\,L\,sin(\alpha)}{l_d}) \tag{3.41}$$

where the turn angle of the circle can be derived from the lateral error as

$$sin(\alpha) = \frac{e_y}{l_d} \tag{3.42}$$

The final control law which expresses the desired steering angle to apply

$$\psi = asin(\frac{2\,L\,e_y}{l_d^2}) \tag{3.43}$$

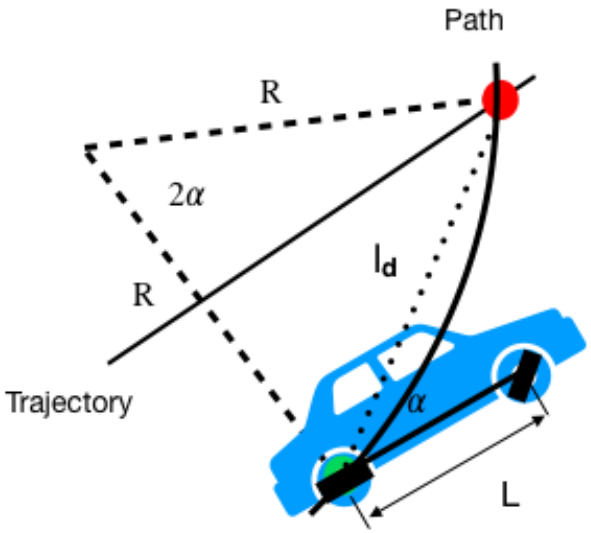


Figure 3.3: The geometry of Pure Pursuit Controller

The look-ahead distance is an arbitrary parameter that can be chosen, depending on the distance to the desired point. The first approach is to define it as the distance from the current robot position to the reference and saturate the resulting expression.

$$-1 <= \frac{2\,L\,e_y}{l_d^2} <= 1 \tag{3.44}$$

The second possible approach is to use the proportional constant. If we know, the sinus of the desired steering angle and the current steering angle, the proportional control law can be applied

$$\psi = C\,(\frac{2\,L\,e_y}{l_d^2} - sin(\psi_{current})), C > 0 \tag{3.45}$$

In this case, the control law expires the changing rate of the steering angle. That is why for the smoother changing of the steering angle, the constant C should be chosen in the range (0; 1).
The third possible approach is to express the look-ahead distance as the proportional constant K of the front speed v in the form

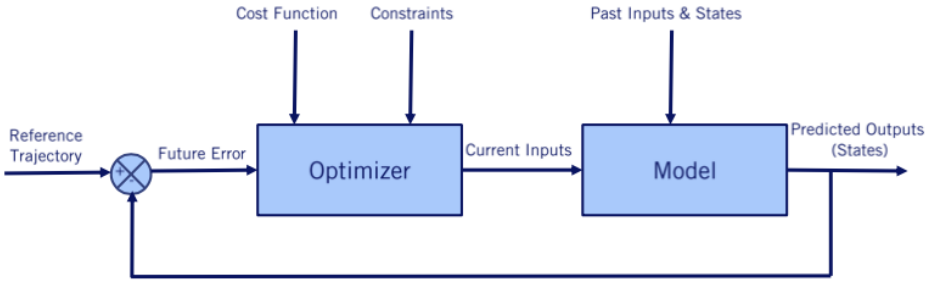
$$\psi = asin(\frac{2\,L\,sin(\alpha)}{K\,v}) \tag{3.46}$$

The proportional controller can be designed but requires the saturation of the fraction divider in the range

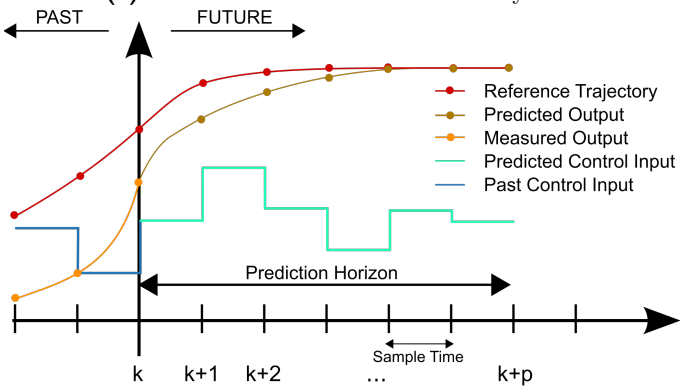
$$K\,v > 2\,L\,sin(\alpha) \implies \begin{cases} K\,v > 2\,L, v > 0 \\ K\,v > -2\,L, v < 0 \end{cases} \tag{3.47}$$

The particular case is when the $\alpha = 0$ when the lateral error is zero. The proposing of the constant K should be designed depending on these parameters. For the testing, only the first two approaches were used. The last one is only the alternative method for proportional control law.
The speed control law was used the same as in the Point to Point Controller from the equation (3.32).

3.2.3 Model Predictive Controller



(a) : Connection of the MPC in the system



(b) : MPC prediction scheme

Figure 3.4: Model predictive controller

Model predictive controller requires the constant access to the plant model of the system to predict the control inputs depending on the cost function to maximize the goal behavior of the system. Using the sampling-based inputs, the controller predicts p number of steps called the prediction horizon. The control horizon is also an important parameter. It defines the number of different control inputs in the first k steps of the prediction. After the application of the p number of sample steps, the input given on the system choice is based on the minimal value of the cost function. For example, if the input is divided on the N samples and it the first k steps, totally N^k possible resulting behavior is predicted, in the next $p - k$ steps the behavior is modeling without increasing the input sample.
Here, we will try to model the control law for the steering angle to enable the

controller to apply the Point to Point controller. All the prediction is based on the cost function in the form.

$$J(x(t), U) = \sum_{j=t}^{t+p-1} \delta \mathbf{x_{j|t}}^T \mathbf{Q} \mathbf{x_{j|t}} + u_{j|t}^T R u_{j|t} \tag{3.48}$$

where

$$\mathbf{x_{j|t}} = \mathbf{e_{robot}} = \begin{bmatrix} e_x \\ e_y \end{bmatrix} - \text{error in position} \tag{3.49}$$

$$u_{j|t} = \psi_j - \psi_t - \text{difference between current and predicted steering angle} \tag{3.50}$$

$\mathbf{Q_{2x2}}$ - diagonal matrix of weights q_1, q_2 and R is a single scalar r that can be determined experimentally.
In every step of the simulation the steering angle is divided on N samples and given to the plant model in the discrete form as

$$x_{t+1} = x_t + \dot{x}\,\Delta t \tag{3.51}$$

$$y_{t+1} = y_t + \dot{y}\,\Delta t \tag{3.52}$$

$$\theta_{t+1} = \theta_t + \dot{\theta}\,\Delta t \tag{3.53}$$

$$\tag{3.54}$$

where the derivation members of the equations are calculated using the constraints of the input and kinematic model of the car-like model from the kinematic equations 3.1.1.
Also, we should mention that the smoother behavior of the system can be achieved using the Adaptive Model Predictive Controller that requires the linearized system in the operational point and based on the linearization the prediction is made. In this thesis, the Adaptive Predictive Controller is not implemented but also can be used as an alternative method.

3.2.4 Pose Controller

The idea of the Pose Controller is to find the control law that will be appropriate for the robot to find the reference of pose $\boldsymbol{x} = [x_{ref}, y_{ref}, \theta_{ref}]$. The reference of the front speed is achieved using the derived speed control law in the equation (3.32).
The control law is based on computing the orientational angular velocity ω_{des} for the unicycle model and applying the transformation between models and proportional control law, and the steering angle is set. The control law of the desired orientational angular velocity has the linear form

$$\omega_{des} = k_1 \alpha + k_2 \beta \tag{3.55}$$

where

- α is the angle to turn based on lateral and longitudinal error as in the equation (3.20) with the constraints of $[-\pi/2; pi/2]$.

- β is the angle between the current and reference orientation as

$$\beta = \theta - \theta_{ref},\ \beta \in [-\pi; \pi) \tag{3.56}$$

The idea is that robot follows to the desired reference point on the straight line that is shifted on the angle β and the final path has the form of the arc. For the stability purposes the constant should be

$$k_1 > 0, k_1 > |k_2|, k_2 < 0 \tag{3.57}$$

which means that the straight-line path following has a significant impact than the orientational difference, but a little bit shifted; otherwise, the turning will be too fast, and stability is broken.
After the computation of the desired angular velocity w_{des} the proportional control law is applied to steering angle change rate in the form

$$\psi = C(\frac{w_{des}\, L}{v} - sin(\psi_{current})),\ C > 0 \tag{3.58}$$

C expressed the proportional change of the current steering angle and desired; the less the constant is, the smoother the controller's behavior. However, in the case of too low gain, the dynamic can not be adapted fast enough to the required changes.

3.3 Motion planning for the Phoenix robot

This thesis aims to develop the library of motion planning maneuvers that would be the most appropriate for the robot in the context of motion planning. All the maneuvers can be divided into non-trajectory and trajectory-based. The non-trajectory require the set of parameters, and the maneuver is applied directly without precomputed trajectory. As an example, we propose the straight-line maneuver described in the subsection 3.3.1.
For the trajectory-based, the idea is to compute the robot's trajectory leading to the final pose and follow it with the path controllers. The trajectory is the sequence of states from the equation 3.15. All the trajectory-based algorithms use time sampling for generating the future states of the robots. For this, we use the sample period T_s, which is chosen experimentally and defined as the time before the robot executes the next state. That time is measured using the clock as the additional ROS node on the actual platform. So, first, we apply the algorithm for generating the trajectory, then save it into .csv format and load it to apply on the robot using the controller.

3.3.1 Straight-line maneuver

Suppose we assume that the path to the desired state is collision-free, which means there are not any obstacles that can lead to a collision between the robot and the environment. In that case, the simplest idea is to use the

Point to Point controllers to generate the path to the target state. The main problem is that the robot final orientation θ_{robot} will be different from the desired θ_{ref} because the Point to Point controller is based on the turning robot toward the point without adjusting the orientation to the reference one. That can be compensated if the turn angle is small

$$|\alpha| < \epsilon, \epsilon \approx 0 \tag{3.59}$$

From the equation this condition is met if the lateral error e_y relative to the robot frame is also small

$$|e_y| < \epsilon \implies |\alpha| < \epsilon, \epsilon \approx 0 \tag{3.60}$$

In that case, only the speed control law from the equation 3.32 from the PTP is applied. For the straight-line maneuver, we define the move distance d_{move} from the init position and the error

$$e_d = |||\mathbf{x_{cur}} - \mathbf{x_{init}}|| - d_{move}| \tag{3.61}$$

- $\mathbf{x_{cur}}$ - current position of the robot.
- $\mathbf{x_{init}}$ - init position of the robot, by default zero in both dimensions.

This type of maneuver can reduce only the longitudinal error e_x and follow the reference state front speed v_{ref}. Also, this maneuver is appliable if the orientation error is minor or ideally zero.
The benefit of this maneuver is the simplicity, and the trajectory is not required for the robot. All that is needed is the reference state and the accuracy of approaching the final point ϵ. Also, the path can be discretized using simple point-to-point and more complicated maneuvers.

Algorithm 1: Straight-line maneuver algorithm

Result: Robot final state $s_f(t)$
Define the robot current state s_{cur};
Define the accuracy, sampling time ϵ, T_s;
Defined distance to move d_{move}, reference speed v_{ref} Define the current distance error $e_d = \infty$;
while $e_d > \epsilon$ **do**
  $e_d = |||\mathbf{x_{cur}} - \mathbf{x_{init}}|| - d_{move}|$;
  v_{set} = SpeedControlLaw(v_{ref}, $d = e_d$);
  s_{cur} = UpdateState(s_{cur}, $[v, \psi]$, T_s);
end
$s_f(t) = s_{cur}$

3.3.2 Rapidly-exploring random tree

The more generalized approach for solving motion planning problems is using the sample-based algorithm Rapidly-exploring random tree(RRT). It is universal and can be appliable when the type of the required maneuver is not known.

Basic algorithm

This algorithm aims to build the tree in the configuration space C that explores it as much as possible. After building the tree using the searching algorithms the path in tree nodes that lead to the optimal solution is found. First, we define the number of tree nodes N_{it} or the number of iterations the algorithm explores the configurational space, then we randomly sample the robot configuration X_{rand}, in our case, the pose $X = [x, y, \theta]$, search for the nearest node that already is in the tree and generate the new node in the direction of the random one using the most appropriate inputs $[v, \psi]$ that minimalize the defined Minkowski distance L_m between nodes in the configuration space as

$$L_m = (|x_1 - x_2|^m + |y_1 - y_2|^m + |\theta_1 - \theta_2)^m)^{1/m} \tag{3.62}$$

For Minkowski distances, the orientational difference should be normalized to defined ranges of $[-\pi; \pi)$. For the algorithm evaluation, we'll use the m=2 distance, also known as the Euclidian distance, defined as

$$L_2 = d(\mathbf{x_1}, \mathbf{x_2}) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + ((\theta_1 - \theta_2)_{[-\pi;\pi)})^2} \tag{3.63}$$

In the ideal case, the distance between the goal state and the closest one converges to zero. Also, after building the tree, every node's costs can be set to one because the time of jumping from one state to another one is the same T_s. The shortest path is the path with minimal time or the number of transition nodes. We assume that in this thesis, we work with collision-free configurational spaces to leave the condition of checking the collisions.

Optimization techniques

The main problem of the basic RRT algorithms is the convergence to the optimal solution that can be optimized using a heuristic approach. The first problem is defining the minimal number of nodes when the algorithms converge to an optimal solution. It is proportional to the distance between the init and final configurations. Changing the value of coefficient K can help to minimalize computational time or search for the path more efficient in the similar configuration space ranges

$$N_{min} = K\, d(\mathbf{x_{init}}, \mathbf{x_{final}}) \tag{3.64}$$

The algorithm is time computational for the ample C-space. One of the optimizations is to define the constant number of iterations N_{extra} when we wait for the convergence in the algorithms; if such is none, then stop the loop to prevent extra computations. So, the basic idea is to iterate through the configuration at least N_{min} and then update the N_{min} using constant K. Also the possible approach is to generatene the m number of different trees with constant N_{min} and choose the tree with the best-found goal node distance.
The next optimization problem is the clever random pose sampling in the

Algorithm 2: Basic RRT algorithm

Result: RRT $G(V, E)$
Define the robot reference, current states s_{ref}, s_{cur};
Define the RRT $G(V, E) = \{s_{cur}\}$;
Define number of iterations N_{it};
Define the accuracy, sampling time ϵ, T_s;
Define the current iteration $it = 0$;
while $it < N_{it}$ **do**
  x_{rand} = RandomPose();
  if X_{rand} *is not collision-free* **then**
    Continue;
  end
  $s_{nearest}$ = GetNearestNode($G(V, E)$, x_{rand});
  s_{new} = GenerateNewNode($s_{nearest}$, x_{rand}, T_s) ; `/* Generate the new node closest to the random one */`
  G(V,E) = Link($s_{nearest}$, s_{new});
  if X_{new} *is close to* X_{ref} *with accuracy* ϵ **then**
    Break;
  end
  $it++$;
end

configuration space . The search space for the motion planning can be defined using as

$$Q(\mathbf{x_{init}}, \mathbf{x_{final}}) = [x_{min}; x_{max}] \times [y_{min}; y_{max}] \times [-\pi; \pi) \tag{3.65}$$

where minimal and maximal values of the x interval(same for y) are chosen with the user preferences of the search space of the algorithm.
The possible solution is to choose a random configuration using the uniform random distribution of all three intervals. However, for the significant intervals or the ample search space, the probability of finding the target configuration using the random samples converges to zero, which minimizes the optimal solution. This can be solved by defining the target point sampling probability P_t , which defines that the final point is generated as the random one in the current iteration. The algorithm generated the random probability. If that one is greater than P_t, then use the uniform random distribution for generating the random point; otherwise, use the target point as the random one. The bigger the probability P_t, the more straight path will be found, which is not optimal in the sense of configurational distance between found goal and real goal states. Also we need to define the step sample value between possible configurations x_i.
The subsequent possible optimization is input optimization. The brute force method divides the inputs v, ψ on some number of samples and finds the pair of samples that generates the new state with the minimal Euclidian distance to the random one. Due to the robot's constraints, inputs do not change

Algorithm 3: Random configuration generation

Result: Random configuration X_{rand}
Define the probability P_t;
Generate the random probability P_{rand};
if $P_{rand} \leq P_t$ **then**
  return X_{target}
end
Define X_{rand};
Define the step *step*;
Define the ranges $ranges_{X_{rand}}$ **for** *configuration* x_i *in* X_{rand} **do**
  x_i = UseUniformDistribution(x_i, $ranges_{x_i}$, *step*)
end
return X_{rand}

at once; that is why the better solution is to use the values of the edges of the input's interval and the zero value. That leaves only the $3^2 = 9$ pairs to search for. If we aim to follow the front speed of reference state v_{ref} using the equation 3.32, then we can use the Point to Point controller principle to adjust speed and only then find the most appropriate steering angle between three possible options. It can be illustrated using the below algorithm.

Algorithm 4: Input optimization

Result: State inputs $[v, \psi]$
Define the robot reference, current states s_{ref}, s_{cur};
Define the random configuration sample X_{rand};
Define coefficient $K > 0$ of proportional controller;
Define possible steering angles array $\psi_{try} = \{-\pi/2, 0, \pi/2\}$;
Calculate distance to the target position d;
if $v_{ref} < 0$ **then**
  $v = v_{ref} - K_v\, d$
else
  $v = v_{ref} + K_v\, d$
end
for ψ_i *in* ψ_{try} **do**
  Choose the one which minimizes the Euclidian distance to the random configuration sample
end

After the generating the tree, any of the search algorithms can be found to find the shortest path to the optimal solution.

3.3.3 Dubin's maneuvers

Basic RRT is an effective algorithm for solving the motion planning problems in the generalized case. In this section, we'll propose the approach for generating the paths to the target state when the type of maneuver is

predefined or can be set using using the motion planing algorithms.

Basic Dubin's curves

The path between two configuration for the planar mobile robot can be derived from different types of Dubin's curves. For the car-like robot model they are represented using the sequence of the three basic motions $\{\mathbf{S}, \mathbf{R}, \mathbf{L}\}$ with the robot inputs $[v_f, \psi]$

- $\mathbf{S} = [v_{set}, 0]$ - go straight.
- $\mathbf{R} = [v_{set}, -\pi/2]$ - turn right with the maximal steering angle,
- $\mathbf{L} = [v_{set}, \pi/2]$ - turn left with the maximal steering angle.

By Dubins the reverse motion is not allowed $v_f >= 0$ and the front speed of the robot is the constant during the all three basic motions, it's value chosen experimentally. The less it's absolute value the more slowly is the robot orientation speed ω. The basic 6 Dubin's primitives are $\{LRL, RLR, LSL, LSR, RSL, RSR\}$.

RRT based on Dubin's curves

The basic concept of Dubin's curves is to calculate the best Dubin's primitive in constant time using the geometric relations between init robot pose and goal pose, which is appliable in case of unicycle model if next conditions are met

- The robot front speed is constant during all three basic motions.
- The angular orientation velocity is changing instantly, with no time constraints.
- The result trajectory is not following the robot's final pose reference front speed.

In case of the car-like model with time constraints the best approach is to search the time of applying the every basic motion. In case of three basic motions and the constant sampling time T_s, the target is to find the three numbers of iterations $n_{1,2,3}$ that form the minimal number of curve iterations in the form

$$n_1 + n_2 + n_3 = N_{min}, n_{1,2,3} > 0 \tag{3.66}$$

The idea is iteratively find the iterations $n_{1,2,3}$ that lead to the most optimal solution for every possible curve and choose the best between them. N_{min} is heuristically chosen parameter that represents the minimal number of iterations for period T_s. We start to search from N_{min} and stop when the founded goal state is close enough to the real one or when the there is no update of the founded goal state for the some number of previous iterations. From the equation (3.66) $N_{min} >= 3$, $N_{max} = \infty$. To prevent searching for too many iterations, we can define the stop counter n_{stop}, which defines if the

better state was not found during the n_{stop} iterations break the loop.
If we also want the robot to follow the reference front speed, we'll define the last basic primitive of Dubin's curve is the Straight maneuver with the reference speed $S(v_{ref})$ when the robot continuously alligns the steering angle to zero and moving straight with the reference speed. We'll define the iteration number n_4 , which is calculated after establishing $n_{1,2,3}$ in the while loop with the condition of setting steering angle to zero or exceeding some n_{4max} iterations. The result maneuvers are derived from the Dubin's curves as

$$\{LRLS(v_{ref}), RLRS(v_{ref}), LSLS(v_{ref}), LSRS(v_{ref}), RSLS(v_{ref}), RSRS(v_{ref})\} \tag{3.67}$$

Also there were added two maneuvers for robot's continus turn and inspired by left and right turn of the vehicle

$$\{SLS(v_{ref}), \{SRS(v_{ref})\} \tag{3.68}$$

For them only $n_{1,2}$ are required and calculated, that implies from the equation (3.66) $n_3 = 0$. Totally we have 8 different maneuvers. We assume that the search space is also collision-free, so we don't need to check the found current state of the robot after applying the maneuver.

Algorithm 5: Dubin's curves RRT

Result: Iterations $n_{1-4}, maneuvertype$
Define the robot reference, current states s_{ref}, s_{cur};
Define the interval edges of iteration N_{min}, N_{max};
Define n_{stop} ;
Define distance to goal as infinity ;
Set update counter = 0 ;
Define the front speed for maneuver v_m, period dT;
Define the n_{4max} ;
for N *in* $[N_{min}, N_{max}]$ **do**
  if *update counter* == n_{stop} **then**
    break;
  end
  for n_1 *in* $[1; N]$ **do**
    for $n_2 in [1; N]$ **do**
      $n_3 = N - n_1 - n_3$;
      if $n_3 < 0$ **then**
        break;
      end
    end
    for *maneuver in maneuver types* **do**
      current state = maneuver(n_1, n_2, n_3, v_m, dT)
    end
    $n_4 = 0$;
    while $|\psi| < 0$ *and* $n_4 < n_{4max}$ **do**
      current state = GoStraight(n_4, v_{ref}, dT);
      $n_4 + +$;
    end
    if n_4 == n_{4max} **then**
      continue;
    end
    if *current state is closer to goal state* **then**
      remember iterations, distance to goal, maneuver type ;
      update counter = 0 ;
    end
    update counter ++;
  end
end

Chapter 4

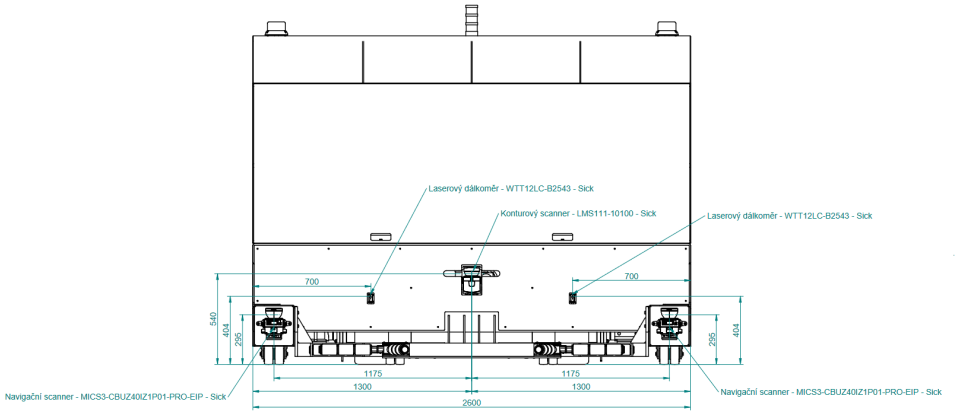
Tools, software and robot system description

This chapter introduces the tools and software used in this thesis for off robot testing and simulation, presents the hardware, construction, and software used on the robot without mathematical description.

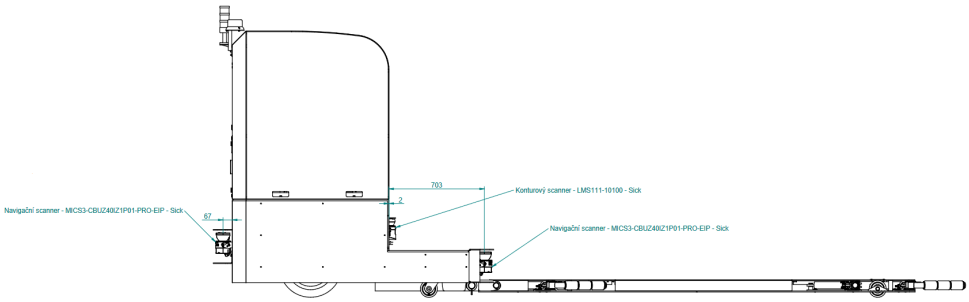
4.1 Robot hardware and software, proportions

Figure 4.1: The Phoenix robot with the docked car

Phoenix robot shown on the figure 4.1 is equipped with a control unit that uses the code written with ROS C++, current the primary source is the Phoenix driver. The robot is controlled with the inputs from the equation 3.8 published to the ROS node *cmd_vel* and receives the information from about current position and orientation from the publisher *odom*. The user can connect to the robot's local network and run all the required ROS nodes. The user also can use the joysticks for the robot moving or stopping in emergency cases. Among the sensors, the robot has the Lidar scanners and camera for navigation purposes. Also, the camera enables to display of the world around the robot. The robot has 5 Lidars: two on the sides, two on the front and one on the back shown on the figure 4.2. Another sensor is the



(a) : Front view



(b) : Back view

Figure 4.2: The Phoenix robot sensors

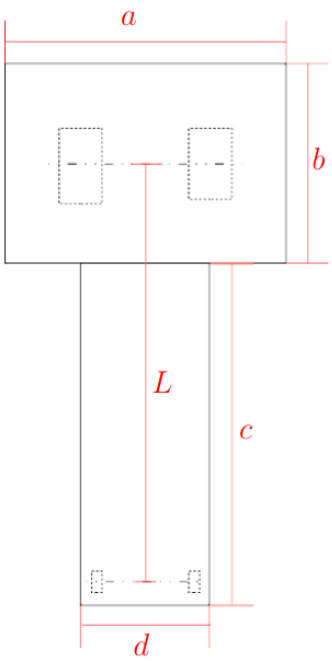
GPS installed on the roof of the robot, which gives the opportunity of the global localization.
The robot shape is quite conservative and visually consists of the box and the car's dock plane. It's proportions are presented on the figure 4.3b. The robot's dynamical properties were not mentioned because they cannot be established precisely. Experimentally, the center of the mass determined in the middle of the docking plane. However, the most significant interest has the kinematic of the robot, which is described in Chapter 3.

4.2 Simulation tools

The simulation and testing of the proposed algorithms were divided into two parts. The testing of the algorithm's performance was conducted in `MATLAB`. Further implementation, with the source codes used on the real robot, was performed in `ROS` with the simulation in `Gazebo`.

Proportion	Length[m]
a	2.6
b	1.5
c	3.5
d	2.0
L	4.18

(a) : Shapes of the Phoenix robot.



(b) : The robot's projection

Figure 4.3: The Phoenix sizes [24]

4.2.1 Simulation in MATLAB

MATLAB is an effective tool for projecting the required algorithms due to the extensive built-in tools and simple programming language. All the proposed controllers were created using this source, and further were adapted for the actual platform. The robot mathematical model that is described in Chapter 3 was implemented in MATLAB function `SteerBot()`same as the one which is used for the real robot but with the slight difference that on the actual platform, all the simulation is also discrete in time. All the time steps for derivation or integration are performed using the clock, whenever in MATLAB the preference was given to strict timesteps. $T_s = 0.05[s]$.
All the simulation used for controllers proposing is quite primitive and performed in the simple while loop when the outputs from controllers are fed to inputs of the system. The controller, the MATLAB function, takes as the input the current state of the robot, described in Chapter 3, and the reference, which is the pose, position, or state of the trajectory depending on the controller type.
MATLAB was also chosen because it allows fast graph plotting and controller evaluation which would take more time if the preference to Gazebo simulator had been given. After all, the ROS requires extra time for compiling a large number of packages in the Phoenix driver.

4.2.2 ROS

ROS(Robot Operating System)is an effective tool for multithreading operation of the robot's system used for the simulation and the real robot. Although MATLAB is effective for projecting the algorithms, seeing the controller's behavior on the real Phoenix manipulator can be observed only with the usage of ROS phoenix driver. All the packages in this driver are compliable with ROS kinetic and melodic. In this thesis, all the simulation tests were performed in ROS Melodic for Ubuntu 18.04. and Gazebo version 9.

4.2.3 Gazebo simulator

The Phoenix team has created a simple robot model in an `SDF` file format which can be imported to the Gazebo simulator. But the implemented robot has some disadvantages in simulation

- It can't be set to the required pose manually in the simulator; it initially spawns in the origin of the world coordinate frame, which is caused by the requirement of the BearNav navigation system on the real robot.
- The robot collisions cannot be simulated due to the undefined dynamic parameters of the robot. It is impossible to follow the robot's behavior at the interaction with the obstacles or other objects.
- The model is static, and it is not possible to visually see the changing of the inputs on the robot wheels.

However, the robot kinematic and the behavior of all the packages of the Phoenix driver can be simulated similarly to the actual behavior of the robot.

4.2.4 Environment, Mesh

As mentioned, the robot works on the Skoda parking. Using the Lidar sensors installed on the robot, the Phoenix team followed the path created the point Cloud of the parking.
In this thesis, using the `Open3D` module and `Ball-Pivoting` algorithm in Python [21] different types of mesh were created. The parameters of mesh are in the table above.

Number of measure points	Ball radius constant	Size[MB]
$7 \cdot 10^6$	3	52
$7 \cdot 10^6$	5	49
$20 \cdot 10^6$	3	150
$20 \cdot 10^6$	5	140

Table 4.1: Properties of mesh

The resulting mesh of 1ml triangles was parsed manually in Blender to delete

the empty holes and saved the .pynb file, which was used for exporting to Gazebo. The resulting computational load was relatively high in Gazebo, and loading to simulator the required meshed required about 5[s], which was not acceptable for further testing and evaluation.

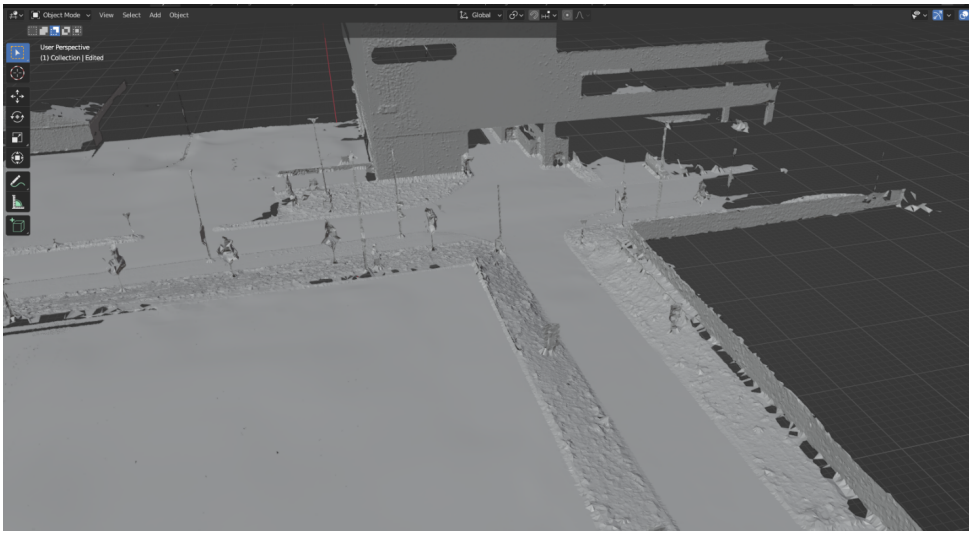
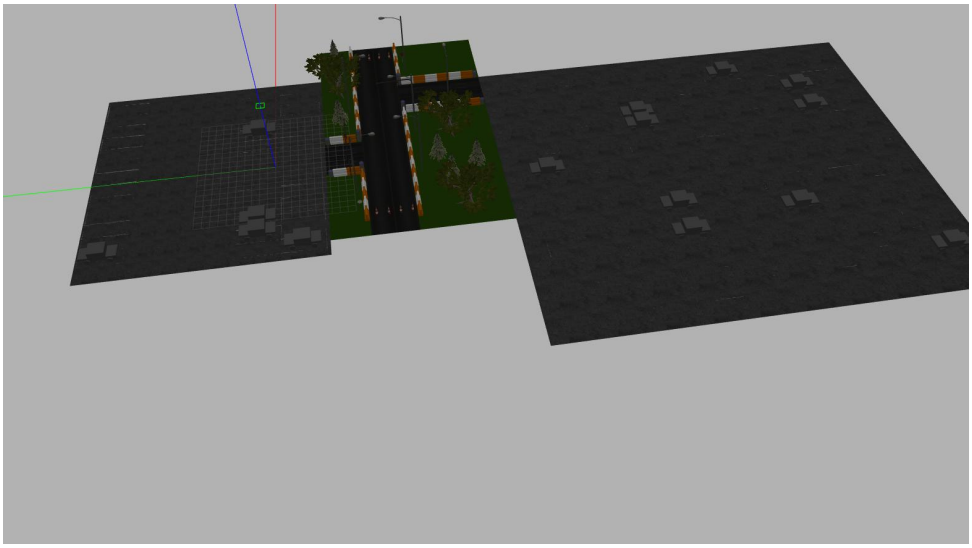
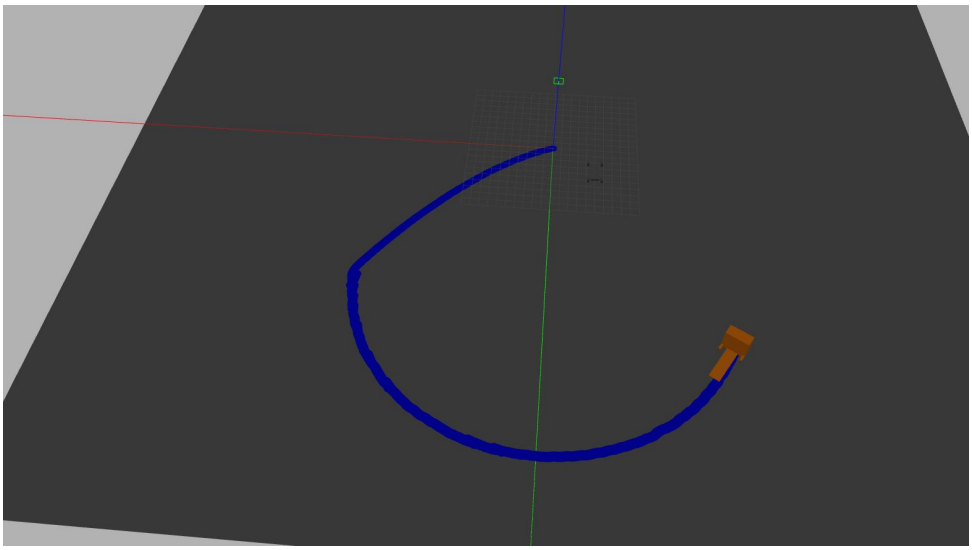


Figure 4.4: The example of parsed mesh

For visualizational purposes was created a simple turn part of the robot path from the basic geometry objects from the Gazebo database [22]. In the project was created a package `gazebo_path_plugin` for random car spawning on the parking lots. Its description is available in Appendix A.1. Briefly, this package enables to spawn N number of plains with the required number of parking lots. It enables the user to spawn the cars randomly by the number of parking lots or the specific required area shown on the fgire 4.5a. This approach can be used for testing the loading maneuvers on the parking. Only for visualization purposes, the package was created for the programmer to see the path in the simulator shown on the figure 4.5b, which the robot was moving. With the combination of these packages, the user can see the possible collisions in the simulator. However, the testing will use the empty world for faster loading and computational stability.



(a) : Example of custom world with spawned parking lots



(b) : Path visualization node in Gazebo

4.2.5 RVIZ

In this thesis, the process of planning methods can be easily visualized in ROS using the RVIZ, which allows visualizing the RRT as the markers array in the configurational space where z-coordinate is equal to the robot's orientation θ, the red marker is the required goal state. The example of generated tree is shown on the figure 5.13.

Chapter 5

Experiments

This chapter introduces the experiments and their evaluation for the proposed algorithms of path following and planning in the simulation and on the real platform.

5.1 Controllers

The evaluation of the path control efficiency is based on the following parameters

- The configurational distance described in the equation (3.63) between goal pose and approached one using the control law. In all presented controllers, except the Pose controller, we will take into account only the positional distance because they do not aim to follow the robot's target orientation. For most of the experiments, we use the maximal configurational distance d_{max} when we consider that robot approached the goal pose.

- The speed deviation between the real and the reference one in the near target state.

The controller with the best behavior minimize the described parameters. In the set of experiments, we'll consider that the initial state from the equation (3.15) of the robot is always zero for all dimensions.

5.1.1 Constant tuning for speed control law

All the proposed controllers use the same control law for the front speed input from the equation (3.32). This section will propose the proportional constant with the maximum impact on the path following. The controller shouldn't have the overshoot and be fast.

Switching to the reference state

To test how effective robot follows the reference in x-coordinate we'll define the reference state with $x_{ref} = 10$, $v_{ref} = 0$ and zeros in other dimensions.

The following state was chosen because we consider the proposed positional reference is enough to reduce any speed error. In the presented experiment, we'll use the $d_{max} = 0.05[m]$, and when the robot approaches the point, switch to the reference speed of zero and measure the steady position, this approach is based on the trajectory following when we discretize it on the set of the points and follow every separately. The experiments were performed with the following proportional gains

$$K_v = \{5, 1, 0.5, 0.47, 0.25\} \tag{5.1}$$

From the figure 5.1 it can be seen that for the larger proportional gains, the doesn't have enough time to stop on the required position and pass by the reference; for too small gains like 0.25, the robot is approaching the final position too slow which is not very robust for the following trajectory. The best-founded gains are limited to 0.5. Their deviation is smaller than required one $d_{max} = 0.05[m]$.

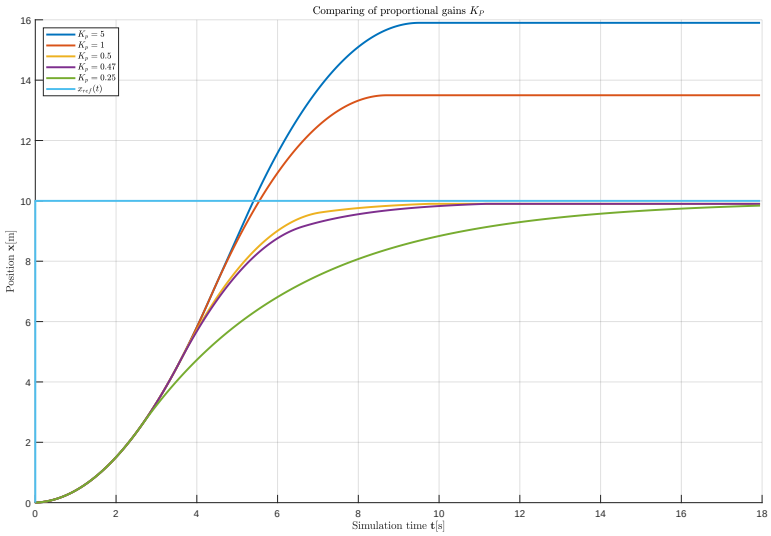


Figure 5.1: Following the zero speed reference state with different proportional gains

Reducing the longitudinal error

The next experiment is similiar to the previous, but we'll change the speed control law to see the overshootings more precicely instead of $\widetilde{sgn(v_{ref})}$ in the control law we'll consider the $sgn(e_x)$. For zero reference speed and reference y-coordinated, the control law simplifies to

$$v_{set} = sgn(e_x)|e_x| = e_x \tag{5.2}$$

The experiment is peformed for the same gains From the figure 5.2 the larger

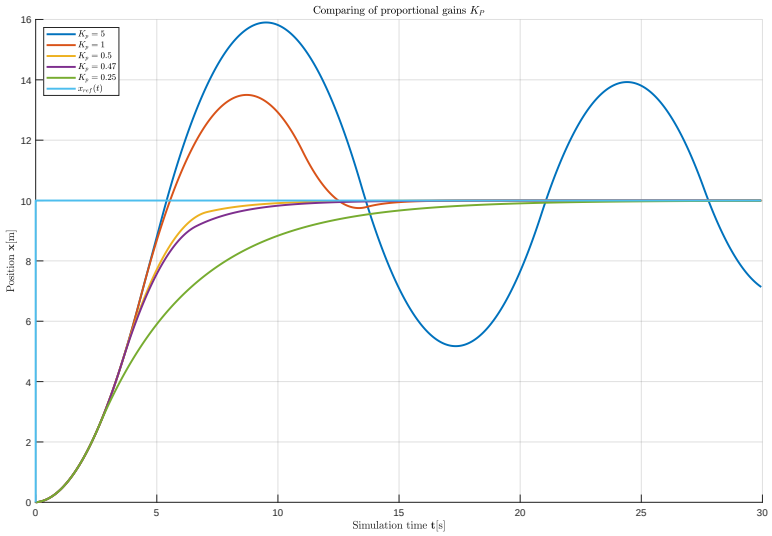


Figure 5.2: Reducing the longitudal error for different proportional gains

gains oscillate and stabilize slower than optimal gains; smaller ones do not have the overshoot but stabilize slower than optimal ones. The behavior of the gains 0.5, 0.47 is quite similar. However, we will choose as the best one the gain $K_v = 0.47$ because its behavior is based on the theoretical background and was calculated from the braking distance from the equation (3.38).

Comparing of speed errors

The last experiment compares the speed errors for different references. We'll divide the reference speeds to 7 samples from [-3; 3] with the step 1 to see whether the controller reduce all the errors when we switch to the position described in the previous experiment. We'll use the proportional gain with the best behaviour.
From the figure 5.3 it can be seen that the robot reduced all the speed errors, which confirms that the controller gain was chosen correctly.

5.1.2 Following the reference poses

The controller parameters were derived experimentally based on the set of the experiments when we generated the reference poses and gradually reduced the satisfactory distance of approaching d_{max} to 0.1[m]. Based on the resulting trajectory, the constants were proposed. In this section, we present the trajectories for poses inspired by the robot's behavior on the crossroad, in some turning cases, shown in the table 5.1. The reference speed v_{ref} in the goal poses was set to 0.

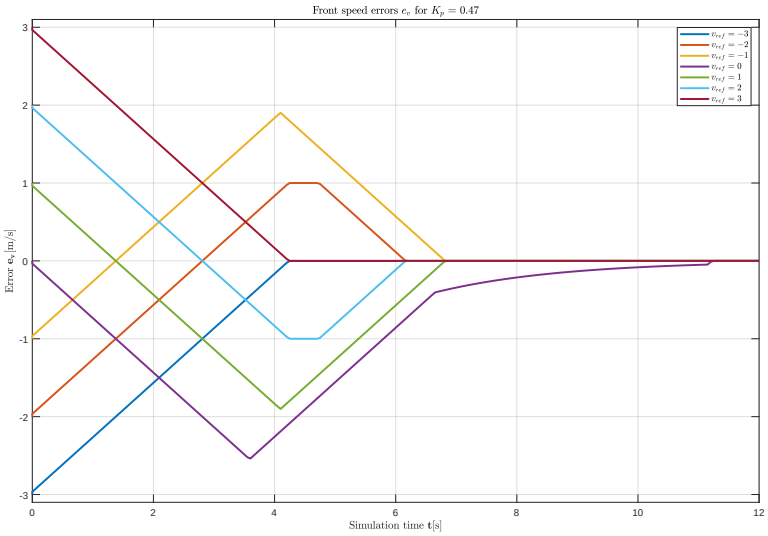


Figure 5.3: Reducing the speed errors for $K_v = 0.47$

Reference pose p_i	Value $[x, y, \theta]$
1	$[5, 5, \pi/2]$,
2	$[0, 5, \pi/2]$
3	$[-5, 5, \pi/2]$
4	$[-5, -5, -\pi/2]$
5	$[0, -5, -\pi/2]$
6	$5, -5, -\pi/2]$

Table 5.1: Reference poses for the experiment

Point To Point

The proportional constant for Point to Point controller $K_v = 0.47$ was derived in the previous experiment. The experiment showed that for smoother turns the proportional constant to the steering control law (3.28) is required, it's value was set to $K_\psi = 0.6$. The final control law is

$$\psi = K_\psi atan2(e_y, |e_x|) \tag{5.3}$$

From the figure 5.4 we can see that the robot successfully followed all the reference poses, but for edge poses like p_2, p_5 it required some extra turns.

Pure Pursuit

For Pure Pursuit controller was used the devoted approach described in the equation(3.45) with the saturation from the equation (3.44). The proportional speed constant was experimentally set to $C = 0.45$

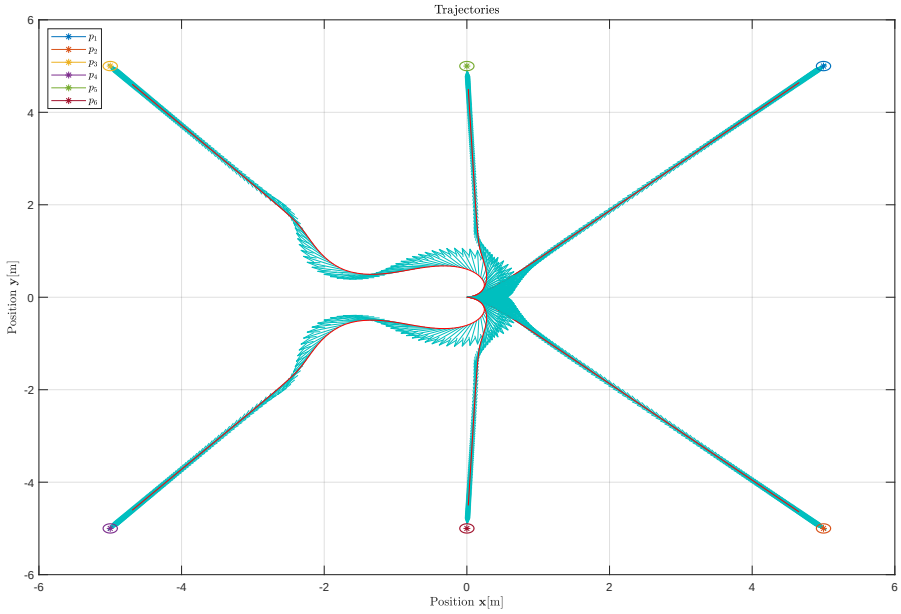


Figure 5.4: Following the reference poses for Point to Point controller

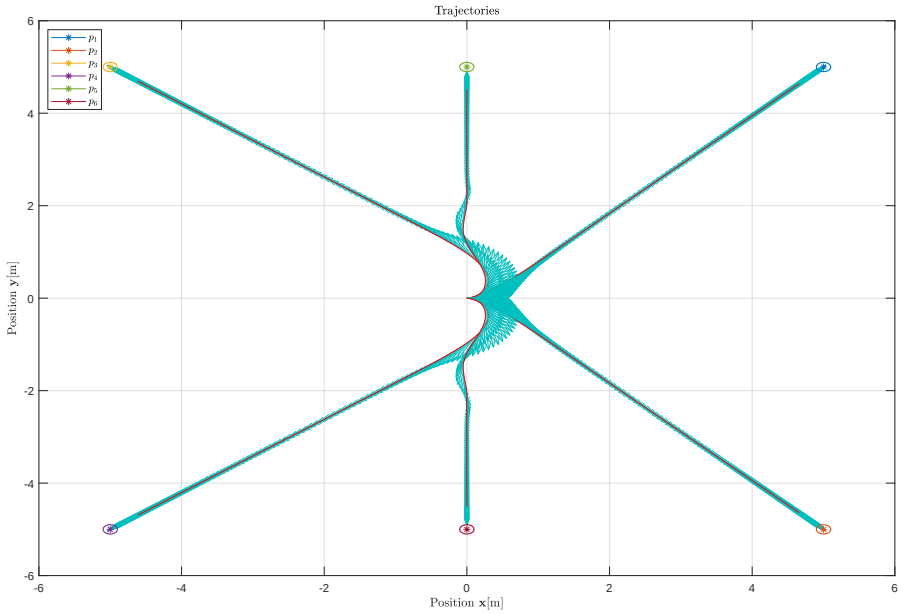


Figure 5.5: Following the reference poses with Pure Pursuit controller

From the figure 5.5 we can see that controller has the preferable behavior, similar to PTP but requires a more significant turn radius.

Model Predictive Controller

MPC aims to predict only the steering input value, the front velocity is calculated inpedentedtly using speed control law in the prediction process. The implemented controller has the time horizon $p = 3$ simulation steps($k = 2$), the weights for position references x, y were $q_1, q_2 = (2,\ 2)$ correspondantly, for steering input ψ the weight is $r = 0.25$ described in subsection 3.2.3. The time step was chosen $T = 0.1[s]$. The number of samples for the steering input was chosen with the step 0.1, which implies the approximately $N = 32$ samples for ranges $[-\pi/2; \pi/2]$ From the figure 5.16 we can see that the robot

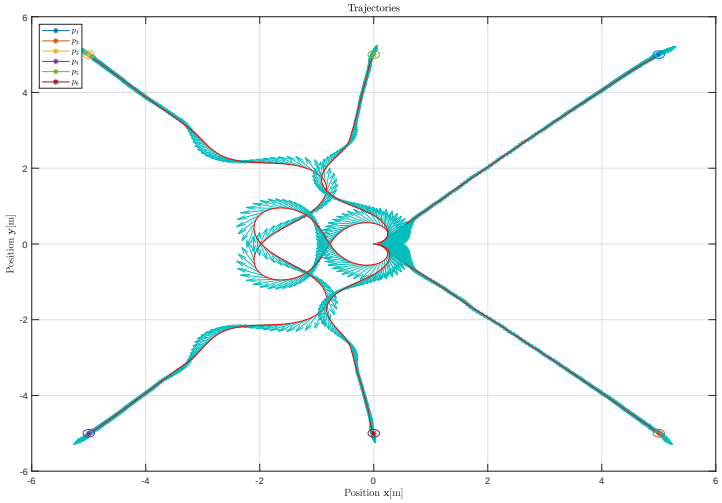


Figure 5.6: Following the reference poses with MPC

following the reference poses but struggles in r poses p_2, p_4 and finds weird trajectories.

Pose Controller

For Pose Controller the proportional constants $k_1 = 1.75$, $k_2 = -0.7$ (3.55). The proportional constant for steering speed $C = 0.25$ (3.58). For all presented points we measured the deviation of orientaional errors e_θ in the final position 5.1. The average absolute value of the errors is

$$\overline{e_\theta} = \frac{1}{6}\sum_{i=1}^{6} |e_{i\theta}| = 0.1390[rad] \tag{5.4}$$

The robot reduces the orientational error quite precisive and follows the reference poses, but the turn radiuses are even more significant than in Pure Pursuit.

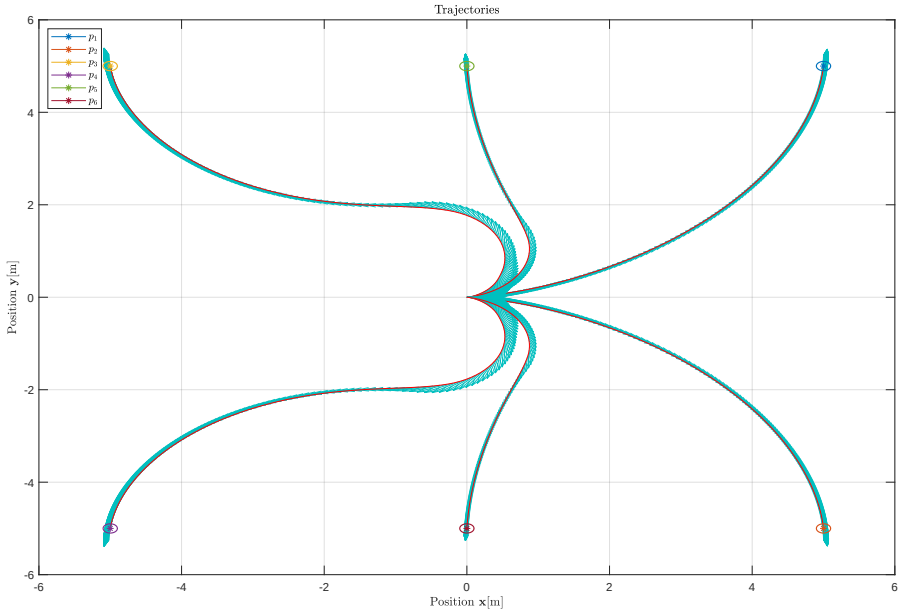


Figure 5.7: Following the reference poses with Pose Controller

Position number	Orientational error $e_\theta[rad]$
1	0.1365
2	-0.1357
3	-0.2097
4	0.2097
5	-0.0711
6	0.0711

Table 5.2: Orientational errors

5.1.3 Following the reference circle

The last experiment is based on the following reference circle trajectory. There are two types of switching between the points in the trajectory.

- `By tolerance`. Define the approaching distance d_{max} and switch to another point when the robot's distance to the current point is less than d_{max}. This approach is very effective in case we want the robot to reduce the initial error, but in case of too small d_{max}, it can loop on one point .

- `By distance`. Switch to another point when the robot's distance to the current one is greater than the previous distance. However, the robot has to start precisely on the first point of the trajectory; otherwise, the trajectory will be followed with the greater average deviation due to the specific architecture of the controller.

First, we'll use the stop on the first approach. We'll generate circle with N_s samples with radius $r = 10$ and center in the beginning of the world frame. The robot's orientatioanl speed is constant and equal to $w = \frac{2\pi}{N_s}$ Experimentally, it was derived that the maximal approach distance d_{max} is approximately equal to the positional distance between samples.

$$d_{max} \approx \frac{l_{circle}}{N_s} = \frac{2\,\pi\,r}{N_s} \tag{5.5}$$

The deviation from this law leads to the wrong behaviour in the trajectory following.

N_S	d_{max}[m]
100	0.60
400	0.15

Table 5.3: The number of sampling in the circle generation

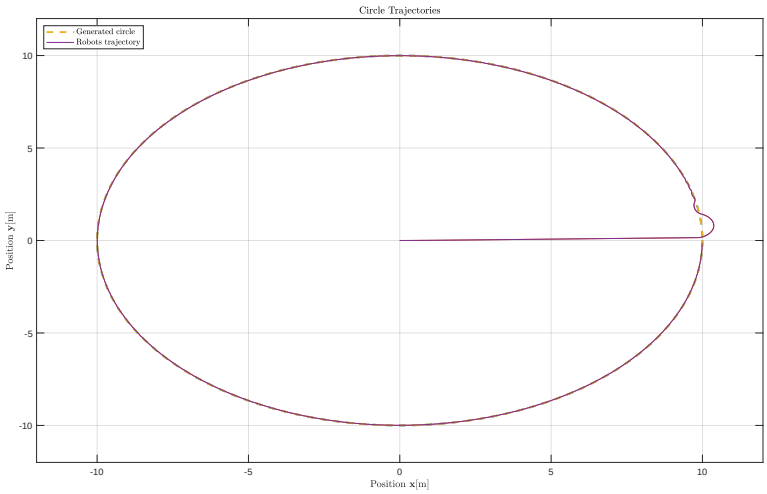
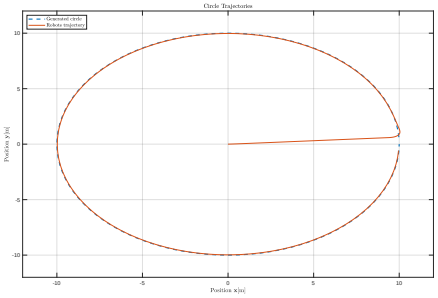
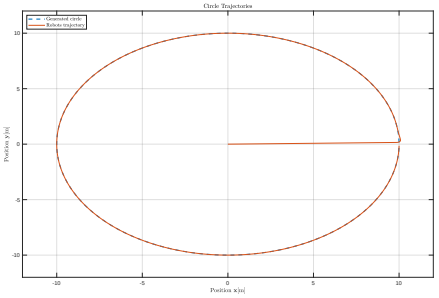


Figure 5.8: Switching points by previous distance for Pure Pursuit, $N_s = 400$

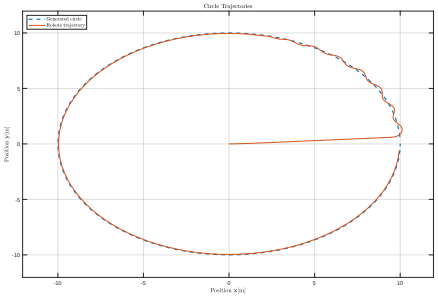
From the figures 5.9 it can be seen that all controllers follow the reference circle precisely. The best behavior was achieved for Point To Point. From the figure 5.9d we can see that robot does not stabilize quite precisely, so the experiment with the point switching by distance was applied. The idea is to switch points by previous distance when the robot finds the first point of the trajectory. The result is presented on the figure 5.8. We can see that robot followed the circle precisely.



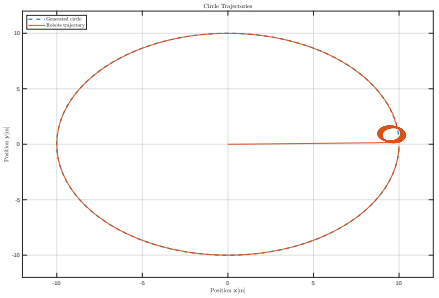
(a) : Point To Point for $N_s = 100$



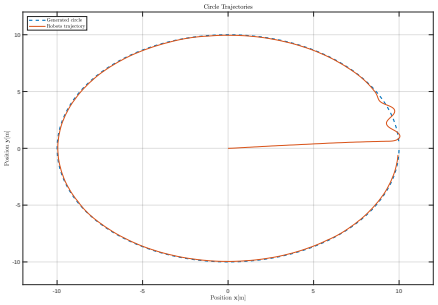
(b) : Point To Point for $N_s = 400$



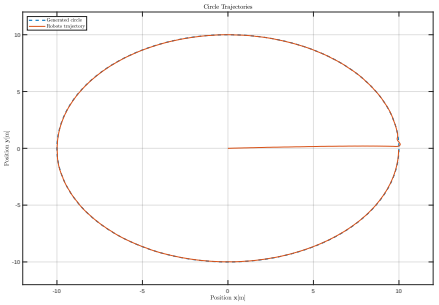
(c) : Pure Pursuit for $N_s = 100$



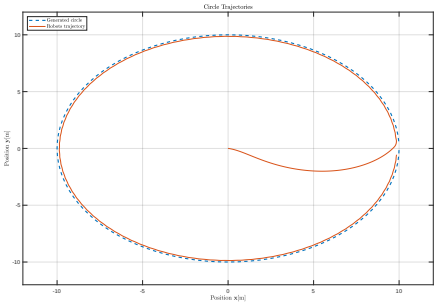
(d) : Pure Pursuit for $N_s = 400$



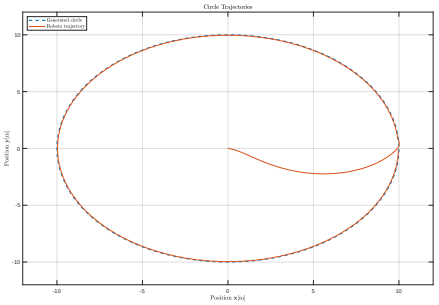
(e) : MPC for $N_s = 100$



(f) : MPC for $N_s = 400$



(g) : MPC for $N_s = 100$



(h) : MPC for $N_s = 400$

Figure 5.9: Circle following with controllers

5.2 Motion planning

The idea behind motion planning is to generate the trajectory as the sequence of points with one of the proposed algorithms and follow it using the path-controllers methods. The sequence of actions in our case is to generate the trajectory, save it in .csv format, and load in the future testing, or choose from already generated ones. As the parameters for maneuvers evaluating, we will use the minimal time of maneuver, accuracy, or configuration distance d_{acc} between the found and real goal poses, the search space in 2D. The presented methods are also part of the maneuver's library.

5.2.1 Straight-line maneuver

The straight-line maneuver is not a classic motion planning problem because it doesn't require the reference trajectory and can be applied straightly without any trajectory controllers. With the use of the proportional constant $K_v = 0.47$ derived in the speed control law, we have measured the simulation time required for the robot's stabilization near reference with some configurational distance d_{acc}. As the experimental set was chosen $d_{move} = 10[m]$, $v_{ref} = 0[m/s]$: "move 10 meters straight and stop". The choice of d_{move} can be explained because will have enough time to get to $v_{max} = 3[m/s]$ and reduce it to zero value.

Accuracy d_{max}	Maneuver time t[s]
0.5	7.25
0.2	7.90
0.1	7.95

Table 5.4: Comparing of straight-line maneuver time

5.2.2 Dubin's curves

For Dubin's curves, the main parameters are the sampling time between states T_s, the number of unchanged iterations n_{stop}, the total number of iterations N_{max}. It's difficult to compare the precise search space because the path generating algorithm depends on the number of iterations for each basic maneuver. Every point is added with the new iteration, which implies that the less the total number of iterations(points) for the maneuver, the less is the search space. Also, the theoretical time of maneuver applying is the sum of iteration multiplied by period T_s. For comparing the result trajectories we'll use the reference pose of $[5, 5, \pi/2]$, planning speed $v_{plan} = 1[m/s]$. This set of experiments will also show how to choose the most optimal trajectory.

As the first experiment, we'll try to define the best sampling period that minimizes the distance from the reference pose d_{max}. Also we set $n_{max} = 10$ and max of aligning iterations to zero steering angle is 20.
From the table 5.5 the most optimal sampling time $T_s = 0.2[s]$ because it has

Sampling period T_s[s]	Accuracy d_{acc}[m]	Maneuver type	Number of Points
0.1	0.682	RLR	141
0.2	0.097	RSL	84
0.5	0.185	RSL	40

Table 5.5: Comparing of the accuracy d_{acc} for the different sampling periods

the minimal value of d_{acc}.
The parameters n_{max} should be correlated depending on trajectory preferences and number of points. For the founded period we'll compare the trajectories for different n_{max} values and choose the one with the minimal value of the search space and the best precise.

Accuracy d_{acc}[m]	Number of Points
0.714	40
0.336	60
0.097	84

Table 5.6: Comparing of different points numbers and the result accuracy of the maneuver

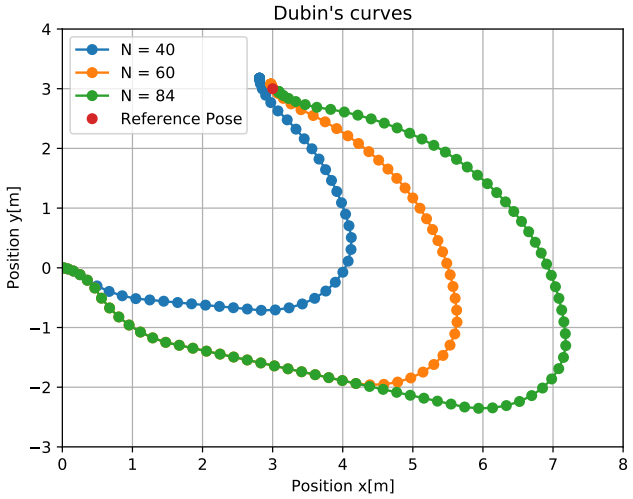


Figure 5.10: Generating Dubin's curves for pose

From the figure 5.10 all the maneuvers are of the same type, RSL, but the more significant number of points give the better accuracy of approximating the reference pose, but also require the bigger working space of the maneuver. The number of stop iterations n_{stop} should be also reckoned, although the algorithm would work without that parameter, but it will search unnecessary loop till N_{max}.

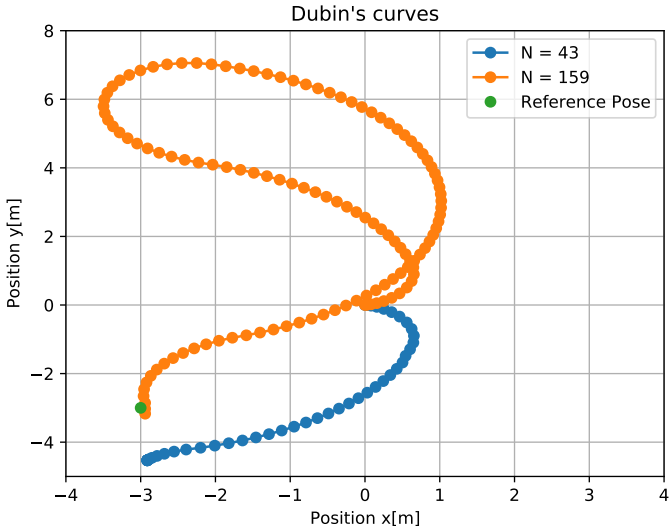


Figure 5.11: Comparing of trajectory points number for [-3, -3, $-\pi/2$]

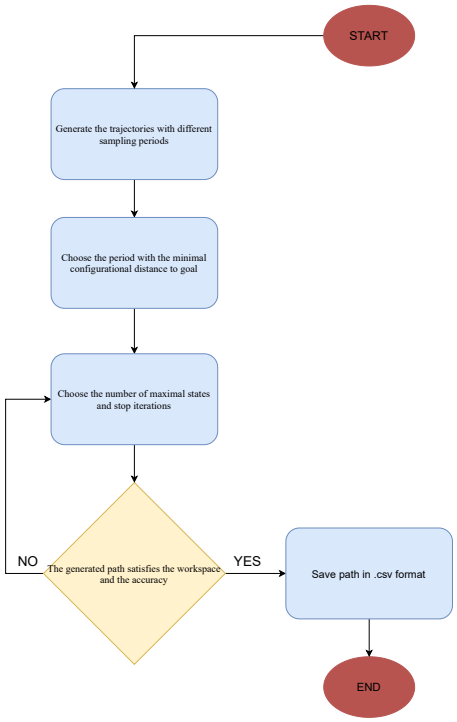


Figure 5.12: Tuning the parameters for Dubin's curves

To illustrate this, we'll perform the same experiment with $N_{max} = 1000$ for pose [-3, -3, $-\pi/2$]. For $n_{stop} = 10$, the algorithms breaks at the iteration $N = 43$ and finds very unprecive solution of $d_{acc} = 1.532$, what doesn't suits us, we start to increase the value to $n_{stop} = 40$ and the algorithm finds the perfect solution of $d_{max} = 0.067$ and $N = 159$ but for bigger number of points also increases the search space which can not be appliable in the narrow spaces 5.11. We propose the following algorithm or the sequence of steps to tune the appropriate parameters for the Dubin's curves RRT 5.12.

5.2.3 Basic RRT

As the RRT is based on random sampling, we will use the principle of generating some number of trees and choose the one with the minimal d_{acc}. Then we apply any search algorithm to find the shortest path to the generated goal pose, the Djiskra algorithm.

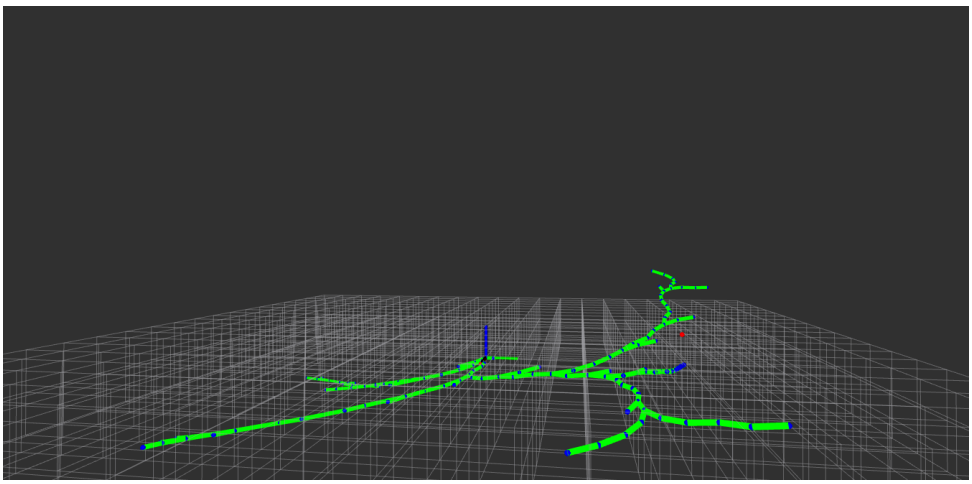


Figure 5.13: Visualization of RRT using RVIZ

For RRT building the sampling period $T_s = 0.5$[s] and the target point sampling probability $P_t = 0.1$ from the subsection 3.3.2 were chosen because comparing to the other ones the accuracy d_{acc} from the goal pose on the same set experiments was minimal. The second parameter of the RRT is the search space or the configurational space we search for when generating the new random state.
To illustrate the trajectories difference and the result accuracy we'll provide the set of experiments to generate the path to goal pose [5, 5, $\pi/2$] for different goal search spaces. The default set is to generate 50 trees with 3000 nodes or points.

From the figure 5.14 and the table 5.7 we can summarize that the RRT algorithm provides more significant precision for bigger search spaces, but it can be challenging to apply in narrow spaces, so we should correlate the search ranges depending on the space preference.

Search space ranges $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$	Accuracy d_{max}[m]	Number of Points
[5, 5]x[5, 5]	$617.8 \cdot 10^{-3}$	30
[0, 10]x[-5, 10]	$4.369 \cdot 10^{-3}$	70
[-10, 10] x[10, 10]	$1.213 \cdot 10^{-3}$	75

Table 5.7: Differences between generated paths for different search spaces

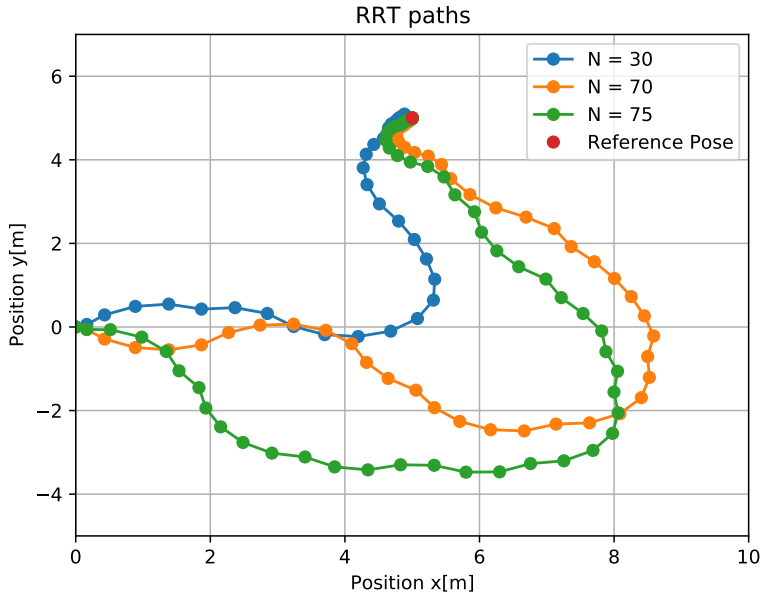


Figure 5.14: The generated path for different search spaces

5.3 Results of the simulation experiments

We can summarize the conducted experiments in the simulation for completing the full experiment on the real platform

5.3.1 Path Controlling

The path controlling problem was successfully solved using the proposed controllers. The most powerful controller that reduces the distance error turned up the simplest Point to Point. To summarize, the flaws of the controllers

- `Point To Point`. Very straightforward, high front speeds require bigger turning radiuses to reduce positional errors.
- `Pure Pursuit`. Requires more time to stabilize, on the small accuracy reference accuracy can cause turning on the circle with constant inputs until following the reference position.

- `Model Predictive Controller`. Computationally demanding sometimes causes unpredicted behavior for different poses.
- `Pose Controller`. Requires a bigger workspace for finding the reference pose without following the pose similar to Point To Point Controllers.

5.3.2 Motion planning

The motion planning problem was solved using the basic RRT and Dubin's curves-based algorithms. They are also the part of motion library

Dubin's curves

Dubin's curves proved to be robust to find the specific trajectory. Tuning the parameters of its generation, we receive the trajectories with the different workspace. We can mention that the workspace is not predefined using this algorithm as a flaw. We need to compare the trajectories with the different points and choose the most appropriate. RRT can generate the path with good accuracy to the goal pose, but the decreasing nodes reduces the accuracy of approximating the pose. Also, this algorithm is very robust when the theoretical trajectory is empirically known, and all are required to smooth the path for the preferences.

Basic RRT

RRT is based on random sampling, so the result of the trajectory accuracy is not predefined. The choice of generating the number of trees and choosing the one with the best accuracy is one of the possible solutions, but due to applying the Djiskta search algorithm for generating the shortest path, it can be computationally long. That's why all the trajectories should be generated in the offline mode of the robot. The parameters tuning is easier compared to Dubin's curves. The workspace is specified.

5.4 Following the reference pose in the narrow space on the real robot

On the real robot we conducted the experiment with the target to generate path to the pose of $[3; 3; \pi/4]$ from init and zero reference speed in the target point.

Trajectory type	Accuracy d_{acc}[m]	Number of Points
Generated	$5.354 \cdot 10^{-3}$	33
Real	$8.72 \cdot 10^{-2}$	33

Table 5.8: Differences between generated paths for different search spaces

Figure 5.15: Result pose of the robot

Based on the simulation experiments results the path was generated using the Basic RRT of ranges [0, 4]x[-1, 4]. To follow the reference trajectory the PTP controller was chosen, the front speed inputs were saturated to [-0.3; 0.3] to follow trajectory more precisely, the switching between points was performed using tolerance method 5.1.3 with the accuracy of 0.1[m].

As the result the robot successfully followed the trajectory with the big precise in the required workspace shown on the figures 5.8 and 5.15.

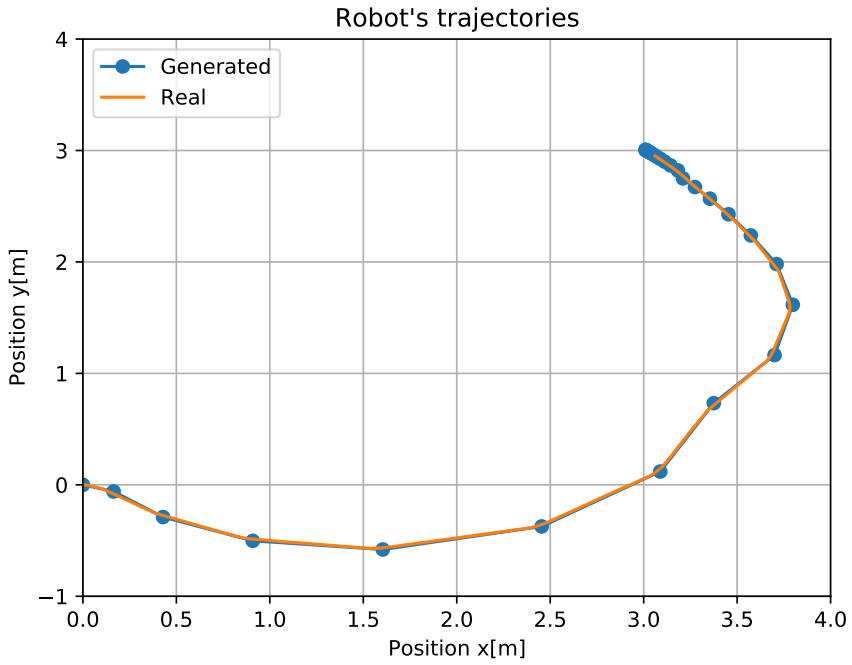


Figure 5.16: Comparing of trajectories for real robot

5.5 Conclusion of the performed experiments

The path-following and motion planning methods were solved in the simulation concerning the proposed requirements. The behavior of the controllers was compared, and the Point To Point as the most successful was performed on the actual platform experiment where we aimed to generate the path in the narrow space with the minimal search space. A basic RRT algorithm completed this task, and the path was successfully followed. Dubin's curves and straight-line maneuvers were tested only in the simulation but are appliable for other motion planning problems.

Chapter 6

Conclusion

This chapter introduces the conclusions of the thesis, present the further work.

6.1 Thesis results and conclusion

The path-following problem based on the theoretical background was successfully solved by applying the path controllers: Point To Point, Pure Pursuit, Model Predictive Controller and Pose Controller. Their behavior was tested on the set of experiments in the simulation of the environment. The motion planning for the collision-free workspace based on the trajectory generation was performed using Dubin's curves and the basic RRT algorithm in the simulation. The ability of easier parameters tuning and simplicity in defining the search-space was the main advantage of choosing RRT for the experiment on the real robot in the narrow space. Point To Point controller was also chosen as the most simple and robust. The task of the real platform experiment was completed. The part of the motion planning solution is the straight-line maneuver, which effectively reduces the distance error and doesn't require the precomputation of the trajectory. This maneuver and the described trajectory generation algorithms consist of the library of maneuvers created for further use on the Phoenix robot.

6.2 Further work

The possible improvement for the following work is to develop the method of collision detecting for the robot because the algorithms are based on the search space. It can be solved by comparing the polygons of the robot's model and the workspace objects on some semantic map. Also, the improvement of RRT can be applied with the RRT star algorithm, which is always finite and provides the shortest path to the goal without any further search algorithms. In RRT generation, we have provided a Dijskra search algorithm that can be replaced with UniCost search, which boosts the speed of finding the shortest path. For Dubin's curves, the most inappropriate part is the parameters tuning which could be solved with the better heuristic of the possible path.

Otherwise, the proposed algorithms and methods provide the solution of the tasks defined in the introduction of this work.

Bibliography

[1] Automatic guided vehicles. (n.d.). AGV Overview. https://www.mhi.org/fundamentals/automatic-guided-vehicles

[2] Robot navigation. (2021, November 27). In Wikipedia. https://en.wikipedia.org/wiki/Robot_navigation

[3] LaValle, S. M. (2006). Comp150-07: Intelligent robotics Notes on C-Space. The Configuration Space. https://www.cs.tufts.edu/comp/150IR/hw/cspace.html

[4] Ravankar A, Ravankar AA, Rawankar A, Hoshino Y. Autonomous and Safe Navigation of Mobile Robots in Vineyard with Smooth Collision Avoidance. Agriculture. 2021; 11(10):954. https://doi.org/10.3390/agriculture11100954

[5] P. Polack, L. -M. Dallen and A. Cord, "Strategy for automated dense parking: how to navigate in narrow lanes*," 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 9196-9202, doi: 10.1109/ICRA40945.2020.9197088.

[6] Ding, Y. (2020, November 16). Simple Understanding of Kinematic Bicycle Model - Yan Ding. Medium. https://dingyan89.medium.com/simple-understanding-of-kinematic-bicycle-model-81cac6420357

[7] Coulter, R. Craig. “Implementation of the Pure Pursuit Path Tracking Algorithm.” (1992).

[8] Reda Ahmad, Bouzid, Ahmed, Vásárhelyi, József. (2020). Model Predictive Control for Automated Vehicle Steering. Acta Polytechnica Hungarica. 17. 163-182. 10.12700/APH.17.7.2020.7.9.

[9] Bower, T. (n.d.). 5.4.2. The Point Forward Steering Controller — Robotics Programming Study Guide. Robotics Programming Study Guide. http://faculty.salina.k-state.edu/tim/robot_prog/MobileBot/Steering/pointFwd.html#the-point-forward-steering-controller

[10] Klančar Gregor, Matko, D., Blazic, Saso. (2005). Mobile Robot Control on a Reference Path. 1343 - 1348. 10.1109/.2005.1467211.

[11] Vieira, F.C., Medeiros, A.A., Alsina, P.J., Araújo, A.P. (2004). Position and Orientation Control of a Two-Wheeled Differentially Driven Nonholonomic Mobile Robot. ICINCO.

[12] Peter Corke. 2013. Robotics, Vision and Control: Fundamental Algorithms in MATLAB (1st. ed.). Springer Publishing Company, Incorporated.

[13] Jean-Claude Latombe. 1991. Robot Motion Planning. Kluwer Academic Publishers, USA.

[14] Kevin M. Lynch and Frank C. Park. 2017. Modern Robotics: Mechanics, Planning, and Control (1st. ed.). Cambridge University Press, USA.

[15] Pin, F., Vaaseur, H.A. (1990). Autonomous Trajectory Generation for Mobile Robots with Non-Holonomic and Steering Angle Constraints. Proceedings of the IEEE International Workshop on Intelligent Motion Control, 1, 295-299.

[16] Lamiraux, Florent, Laumond, Jean-Paul. (2001). Smooth motion planning for car-like vehicles. Robotics and Automation, IEEE Transactions on. 17. 498 - 501. 10.1109/70.954762.

[17] Navigation Toolbox Documentation. (n.d.). MATLAB. `https://www.mathworks.com/help/nav/`

[18] Automated Driving Toolbox Documentation. (n.d.). MATLAB. Retrieved November 1, 2021, from `https://www.mathworks.com/help/driving/`

[19] Sakai A. (n.d.). atsushisakai.github.io. Python Robotics. `https://atsushisakai.github.io/`

[20] P. (n.d.). GitHub - python-control/python-control at 0.8.3. GitHub. `https://github.com/python-control/python-control/tree/0.8.3`

[21] Poux F., PhD. (2021, April 12). Generate 3D meshes from point clouds with python | Towards data Science.`https://towardsdatascience.com/5-step-guide-to-generate-3d-meshes-from-point-clouds-with-python-36bad397d8ba`

[22] O. (n.d.-a). GitHub - osrf/gazebo_models: Model database. Gazebo Database. `https://github.com/osrf/gazebo_models`

[23] Vogel, J. (2021, April 6). Tech Explained: Ackermann Steering Geometry. Racecar Engineering. `https://www.racecar-engineering.com/articles/tech-explained-ackermann-steering-geometry/`

[24] Pěčonková V.(2021). Motion Planning for Autonomous Car Manipulator[Bachelor's thesis, Czech Technical University in Prague]. ČVUT DSpace. https://dspace.cvut.cz/bitstream/handle/10467/96656/F3-BP-2021-Peconkova-Veronika-Peconkova%20Veronika%20-%20BP%20-%20motion_planning_for_autonomous_car_manipulator.pdf?sequence=-1&isAllowed=y

Appendix A

Intro to source codes

The source codes consist of MATLAB and C++ ROS. Matlab provides the path controlling methods in the discrete-time and might be shown for the constant tuning of the proposed controllers.

A.1 ROS platform codes

The source codes implemented for the real robot are provided in the default folder named Path-Planner-For-Phoenix-Environment. You need to add the src folder of your ROS workspace; then, you need to clone from GitHub the actual version of the Phoenix driver and Skoda simulation(private links) to your workspace. After catkin_make your workspace, you are ready to use the packages and nodes of implemented code.

A.1.1 gazebo_path_plugin

Allows generating the parking world with slots without collisions with the real robot. Some of the words have already been generated for visualization purposes. The loading of the world to Gazebo simulator has to be combined with the launch files provided in sim launcher of Skoda simulation.'

A.1.2 trajectory_viewer

Spawns the circles that visualizes the trajectory in the Gazebo, the subscriber to status and odometry publishers from Phoenix ROS driver. 4.5b

A.1.3 basic_maneuvers

The main package is described in the thesis's algorithms and methods. We'll describe the main nodes of these packages

- *applyController*: moves to the specific state using one of the implemented controllers.
- *applyDubins*: applies the specific Dubin's maneuver without any predefined trajectory on the preferable number of iterations for every basic maneuver.

- *generateRRTpath*: generates the RRT, applies Dijisktra search for the shortest path and saves the generated trajectory to a specific folder.
- *generateDubins*: generates the path based on Dubin's curves.
- *followByDistance*: run, when you need the robot follow the generated trajectory by distance method 5.1.3, load .csv trajectory from the specific folder.
- *followByTollerance*: robot follows the trajectory By some tolerance or accuracy the point, specify it and load the trajectory in .csv format.

A.1.4 The .csv format for trajectory

The trajectory is saved as the matrix (n+1)x5, where the columns of the matrix are $[x, y, \theta, \psi, v]$, the first row is skipped and has the string format as [x, y, theta, psi, v]. It can be easily viewed with different tools like Python or MATLAB.

A.1.5 The library of maneuvers

The library is provided with the trajectories and nodes for generetaing them *generateRRTpath*, *generateDubins* where the user can choose the most appropriate one and the nodes of `applyController`, `applyDubins`, *straightLineManeuver* where the user can try the maneuver without the previosuly generateed trajectory.