The Thesis Committee for Parthasarathi Ainampudi Certifies that this is the approved version of the following Thesis:

Box Pushing with a Mobile Robot Using Visual Servoing

APPROVED BY

SUPERVISING COMMITTEE:

Raul G. Longoria, Supervisor

Ashish Deshpande

Box Pushing with a Mobile Robot Using Visual Servoing

by

Parthasarathi Ainampudi, B.Tech

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2019

Dedicated to my family, without whose support, I would not be where I am today.

Box Pushing with a Mobile Robot Using Visual Servoing

Parthasarathi Ainampudi, M.S.E
The University of Texas at Austin, 2019

Supervisor: Raul G. Longoria

Pushing is one of the many ways to manipulate an object and it is especially useful when the object is too big to be gripped by the robot. Previous studies analyzed the problem of pushing polygonal objects into desired poses and proposed open loop pushing algorithms. In the absence of object pose feedback, to avoid slip in contact between the robot and the object, primitives for path planning were computed using conservative estimates for coefficient of friction between the robot and the object. In this work, we experimentally measured the coefficient of friction between the robot and object to compute path planning primitives. We used A* search and RRT* algorithms for path planning. We perform controlled pushing using object pose feedback obtained from a vision system using fiduciary markers. Pushing objects with object pose feedback enables us to confidently operate close to the frictional limits of the system, as the robot can take paths with tighter turns (smaller turning radius) to push the object into the goal pose.

Table of Contents

List of Figures

Chapter 1

Introduction

Humans manipulate objects of varying sizes and shapes without much difficulty. On the other hand, robots struggle even with the basic autonomous manipulation tasks like opening a door handle, opening a bottle cap, etc. We want robots to have the capability to manipulate objects in cluttered environments autonomously. There are two main categories of approaches to solve the problem. We can solve the problem using statistical or learning approaches as can be seen in Levine et al 2018 [4], where they applied model free deep reinforcement learning to solve a high dimensional manipulation problem from scratch in simulation. Another interesting work is done at OpenAI [5], where they perform vision based object reorientation with a dexterous hand.

The second approach to solve manipulation problems is to understand the physics of the process and come up with planning and control algorithms for manipulation, based on the physics knowledge. This is the approach to be taken in this thesis. For simplicity, we'll be focusing on planar pushing. Goyal et al [6] has presented a geometric interpretation of the frictional moment and force acting on a planar sliding object. Lynch and Mason [17] built on Goyal et al [6] and provided the controllability analysis of the pushing process.

They also proposed a path planning approach for planar pushing using a line contact.

There are some approaches that fall in between the above stated categories. One example is the dataset created by Yu et al [24], where they collected high fidelity data from pushing experiments. They then created a probabilistic data-driven model using that data in [1].

A common approach in robotic manipulation is to equip the robot with an end effector (gripper) and take a pick-and-place approach. In this approach, the path planning and grasping are independent of each other because we plan the grasp such that we won't lose grip in the presence of highest expected disturbance forces. The grasp and manipulate approach might not work in cases where the object is too large to be grasped by a gripper or it is too heavy to be lifted. In those cases, we need to use pushing to manipulate the object, which is nonprehensile manipulation.

In this study, we expand on Lynch and Mason's work [17] which proposed open loop algorithms for pushing objects into desired poses. To avoid slip in contact between the robot and the object in the absence of object pose feedback, primitives for path planning were computed using conservative estimates for coefficient of friction between the robot and the object. In this work, we measured the actual coefficient of friction between the pusher and slider to compute path planning primitives. We used A* search and RRT* algorithms for path planning. We perform controlled pushing using object pose feedback obtained from a vision system (Apriltags) using fiduciary markers. Pushing objects with object pose feedback enables us to confidently operate close to the frictional limits of the system, as the robot can take paths with tighter turns (smaller turning radius) to push the object into the goal pose. Open loop manipulation schemes will require us to make conservative estimates for friction coefficients.

There is no experimental data reported in literature to verify the analysis used to compute the path planning primitives in [17]. In this work, we conducted experiments to push objects with a mobile robot using visual servoing. We demonstrate the effectiveness of these principles and algorithms using the pose data collected in pushing experiments.

In Chapter 2, we describe the kinematics of the mobile robot used in the experiments, and present the state space equations used to model the robot. In Chapter 3 the vision based localization system developed for this study is described. We used Apriltags, where localization is done through detection of fiduciary tags in an image. In Chapter 4, the analysis of pushing a polygonal object on a rough surface is discussed and the minimum radius of turning used path planning is derived. In Chapter 5, the planning and trajectory tracking algorithms used in the study are discussed. Finally, Chapter 6 is used to present experimental results, and we conclude the thesis in Chapter 7.

Chapter 2

Kinematics Of The Mobile Robot

2.1 Introduction

We can perform manipulation tasks with different kinds of robots like articulated serial-chain industrial robots, delta robots, soft robots, mobile robots, etc. The dexterity of the manipulation depends on the end effector design and the number of degrees of freedom of the robot.

In this study we consider a pushing task with no grasp. A mobile robot tries to push a box to a desired pose on a 2D plane using a line contact. It is *nonprehensile* manipulation, and the process has non-holonomic constraints. This study adopts a differential drive robot with two driving wheels and one redundant omni-wheel as shown in Figure 2.1. Only planar motion is considered in this work.

2.2 Kinematics Model

The differential drive robot has been studied extensively in the literature and it has standardized models. We will be using the model shown in [22] with slight modifications. This robot has three degrees of freedom (three states). Two states are for the 2D translation and one for the orientation

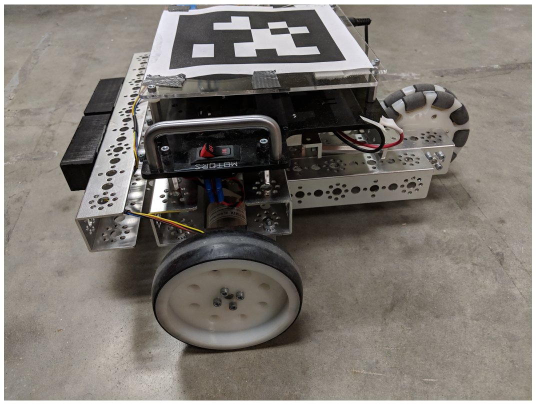


Figure 2.1: Differential drive robot with an omni wheel in the rear.

(along an axis perpendicular to the plane of motion). Let (x_g, y_g) represent the co-ordinates of the center point of the axle of the driving wheels in the *global* inertial frame, whose origin O_g is fixed to the ground. The angle θ is formed between the robot's longitudinal axis and the global axis X_g. The axes X_g, Y_g represent the inertial frame, as shown in Figure 2.2. We will also define another frame at (x_g, y_g) oriented at θ, which will be the local (or body-fixed) frame of the robot with axes X_R, Y_R. We will use the local frame later in Section 5.4 to control the robot.

The robot state is defined as:

$$S = \begin{bmatrix} x_g \\ y_g \\ \theta \end{bmatrix} \tag{2.1}$$

The control inputs to the robot are forward velocity, V, and yaw velocity, ω. Here V is the linear velocity of the center point of the axle of the driving wheels in the ground frame in the direction of X_R. We are assuming that the wheels of the robot are not slipping, which implies velocity along Y_R is 0. The

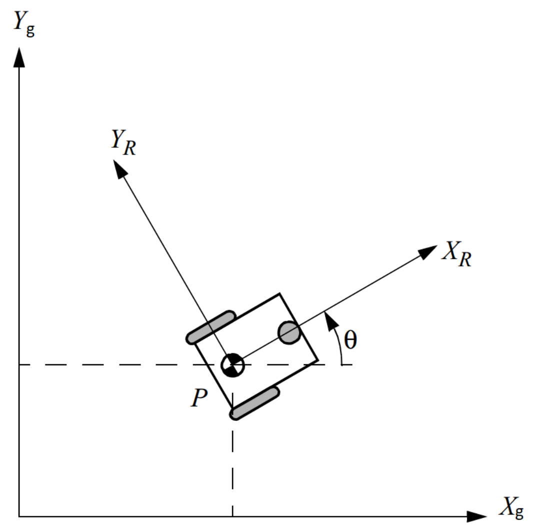


Figure 2.2: Reference frames. Source: [22]

angular velocity of the robot in the ground frame is ω. The conventions are shown in Figure 2.2.

$$\dot{S} = \begin{bmatrix} \dot{x_g} \\ \dot{y_g} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} Vcos(\theta) \\ Vsin(\theta) \\ \omega \end{bmatrix} \tag{2.2}$$

To get the forward kinematics of the robot for a given trajectory of V and ω we need to integrate equation 2.2 with time. We are assuming the wheel slip is negligible. Under that assumption, for a given V and ω, the robot should execute a circular motion with radius R.

$$R = V/\omega \tag{2.3}$$

Figure 2.3 shows experimental data that confirms our kinematics model, where the robot executes a circular motion with radius 0.633m when $V = 0.15$ and

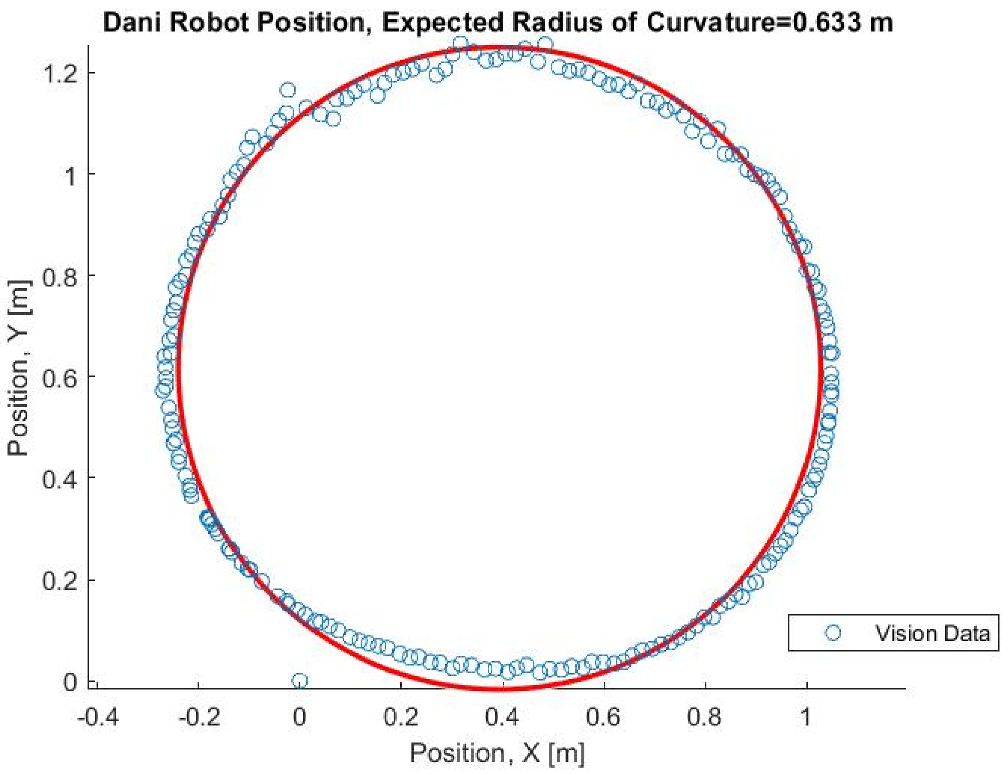


Figure 2.3: Mobile robot executing a circular motion to validate the kinematic model. The data shown is the center point of the robot captured from the vision system.

$\omega = 0.24$. It is not a perfect circle as shown by the experimental data in Figure 2.3 because there are disturbances like floor not being flat, vision measurement errors, etc. We will perform trajectory tracking in Section 5.4 and will use equation 2.3 to more precisely achieve a desired path.

2.3 Controlling The Wheel Speeds

As we have seen before, V and ω are the primary control inputs for the robot. We can't directly use those values to drive the robot. We need to compute the right and left wheel angular velocities from V and ω, which are ω_r and ω_l, respectively. These can be used to command the motor speeds. The width of the robot is b and the radius of the wheels of the robot is R_{wheel}. The following relations hold when there is no slip:

$$\begin{aligned} V &= \frac{R_{wheel}}{2}(\omega_r + \omega_l) \\ \omega &= \frac{R_{wheel}}{b}(\omega_r - \omega_l) \end{aligned} \tag{2.4}$$

From equations 2.4, we can determine desired wheel angular velocities ω_r and ω_l as follows:

$$\begin{aligned} \omega_l &= (V - \omega b/2)/R_{wheel} \\ \omega_r &= (V + \omega b/2)/R_{wheel} \end{aligned} \tag{2.5}$$

We need to ensure that both sides of the robot are equally powered. If we give a command with positive V and $\omega = 0$, then the wheel velocities are identical. Now, if motors on both wheels don't perform identically and if we get $\omega_r \neq \omega_l$ at some point, then from equation 2.4, we can see that we will end up with a non-zero ω, which is not what was commanded.

The kinematic model of the robot can be used to more effectively control the robot trajectory, as will be described in Chapter 5. The next step is to localize the robot using computer vision, so we can perform *visual servoing.*

Chapter 3

Localizing The Robot

3.1 Introduction

For autonomous navigation of robots, we need to know where the robot is in an environment. The goal of localization is to establish the robot's location and orientation in a world frame of reference. In general when we want a robot to reach a goal location, we need to do three things [29]:

- Self-localization
- Path Planning
- Map Update

Localization needs to be done before path planning because we have to know the robot's current location and the goal location. Map update operation is done in parallel as the robot keeps discovering new obstacles and gets fresh sensor data about the environment. For a robot to understand its surroundings through computer vision, it needs to understand the visual features in a scene and have some prior knowledge about those features. In this project we will localize a robot using (artificial) visual features in its surroundings.

3.2 Fiduciary Markers

In the field of machine perception, naturally occurring features in vision are used to detect objects in a scene and localize them. This is generally cumbersome and counter productive when perception is not the central objective. It is much more simpler if we could use artificial features which are much more easier to detect and robust to disturbances. Many different ways to create such artificial visual features are proposed as shown in Figure 3.1. In this project we'll be using Apriltags, which are fiduciary markers proposed by Edwin Olson [20]. Their example applications are seen in Figure 3.9.

3.3 Camera Calibration

Our ultimate goal is to relate a point in real world 3D space to a pixel in the image. Also, the cheap web cameras are pinhole cameras and introduce a lot of distortion. We also want to able to remove the distortion in the image. There are two factors affecting that relation, Camera intrinsics and Camera extrinsics. Intrinsic parameters deal with the camera's internal characteristics like focal length, skew, distortion and image center. Extrinsics deal with the camera's orientation and location in the real world. Once we estimate the internals, we can start computing the extrinsics and make sense of the scene in euclidean space. Most of the standard camera calibration libraries (OpenCV, Matlab, etc) fundamentally depend on the algorithm in Zhang's paper [25].

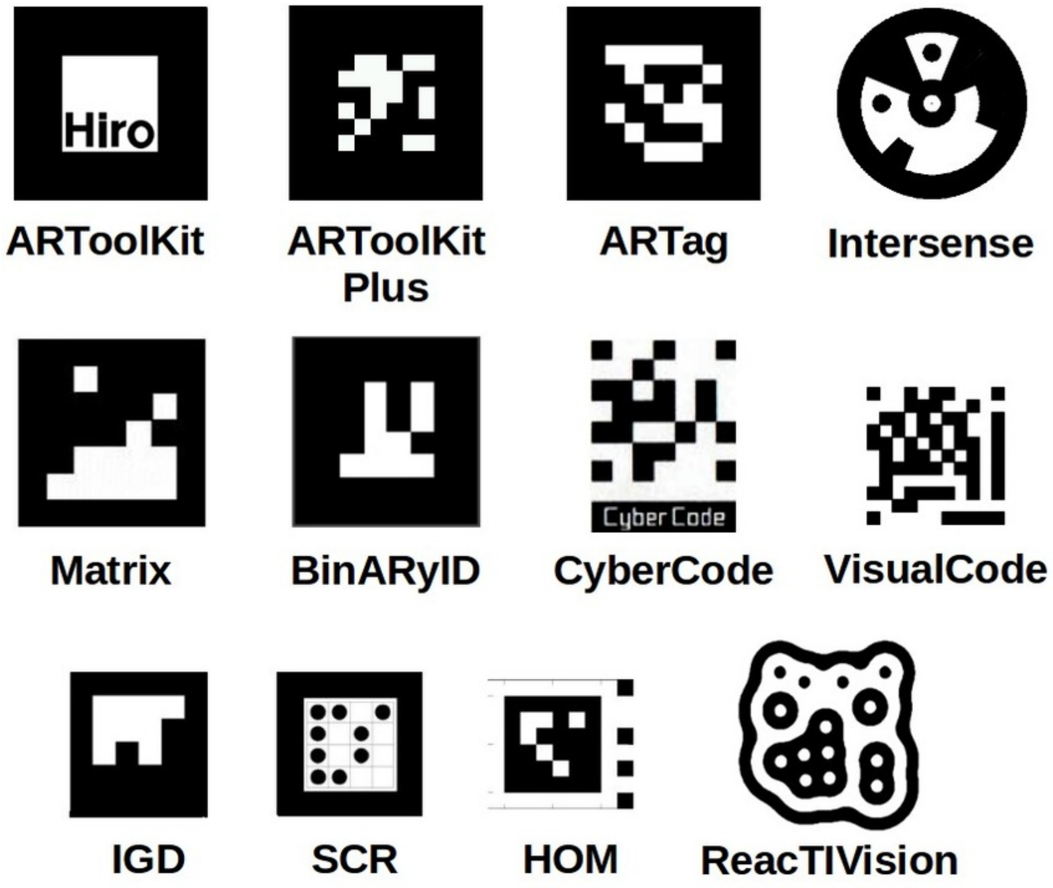


Figure 3.1: Different types of fiduciary markers proposed in literature.

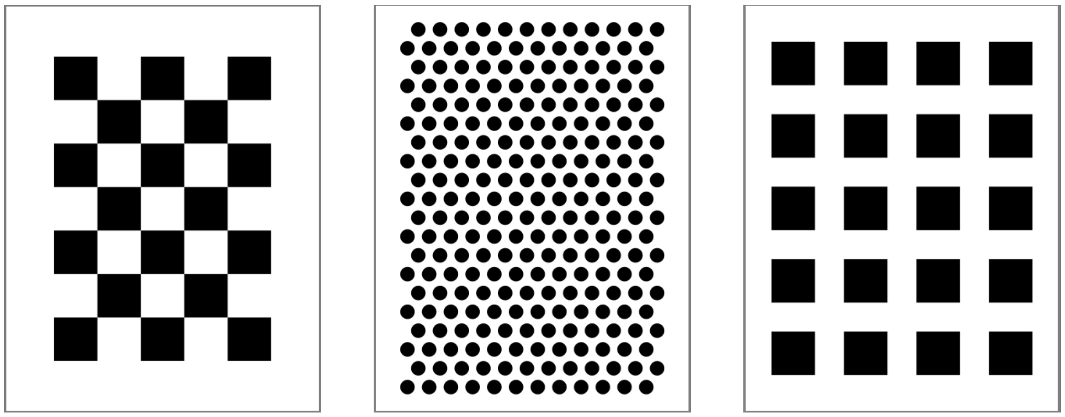


Figure 3.2: Different camera calibration targets. From left to right: chessboard, hexagonal circles grid and square grid

To estimate the camera parameters, we need some known relative distances in the real world so that we could correlate those points in the image. The are many target patterns which give us known distances between different points in the pattern(Figure 3.2) and chessboard pattern is the one which we will be using.

3.3.1 Image Distortion

In general all pin hole cameras introduce some distortion in the image and the cheap webcams do it much more. An example of barrel distortion in an image is shown in Figure 3.3 (left). The board and the ceiling lines are straight in reality but they appear curve in the image. Notice that the same lines appear straight in the rectified image.

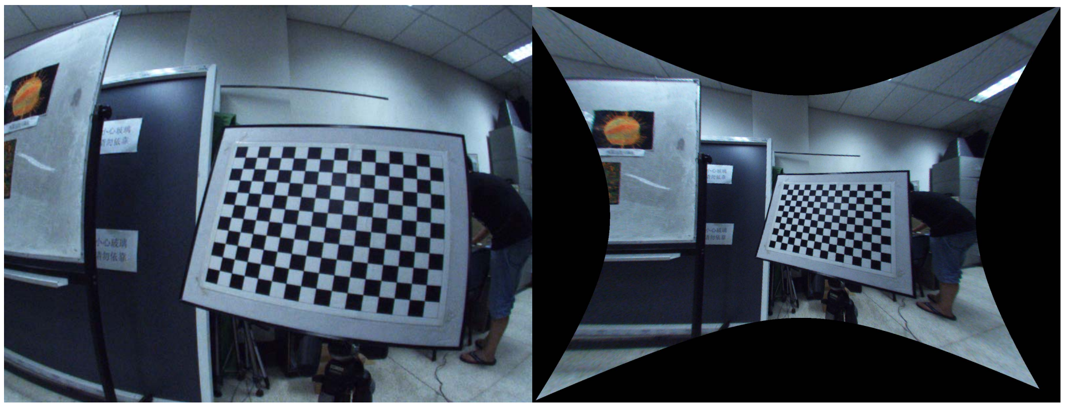


Figure 3.3: Example of barrel image distortion. Notice that the curved lines in the left image and straight lines in the rectified image. Source: [33]

The cameras that we used in this project don't perform as badly as shown in Figure 3.3 but since we need high precision tracking (˜1cm accuracy at 2m distance) we have to rectify our images even for small distortions. There are different types of distortions (Figure 3.4) and We'll be observing barrel distortion in most of the cases.

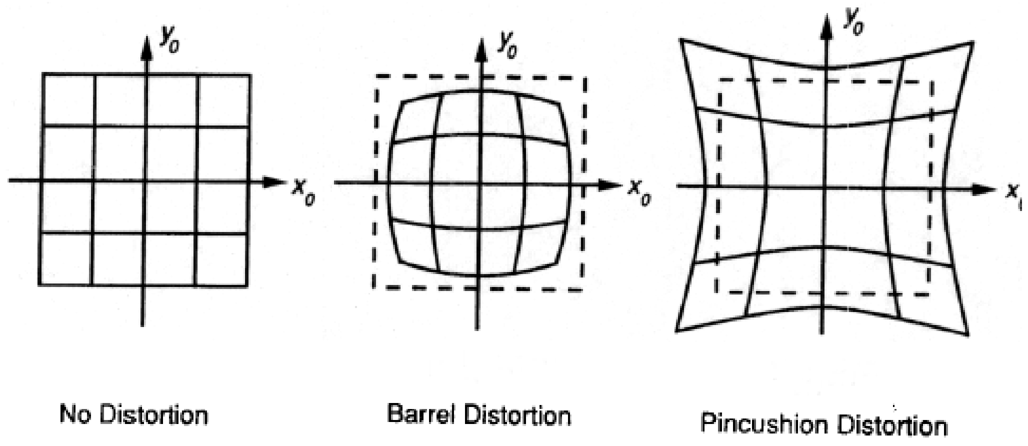


Figure 3.4: Different types of distortion. Source: [28]

3.3.2 Parameters

If we represent the pixel co-ordinates in an image with (x,y) (see Figure 3.4) and r is the radial distance from the center, then we can model radial distortion as follows.

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.1}$$

$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.2}$$

There is also tangential distortion due to the fact that image taking lens is not aligned perfectly parallel to the imaging plane. So some areas in image may look nearer than expected. This distortion can be modeled as follows.

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{3.3}$$

$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \tag{3.4}$$

From the above equations, we can see that we need 5 parameters to model distortion and rectify it later

$$Distortion\ coefficients = [k_1\ k_2\ p_1\ p_2\ p_3]$$

In addition to this, we need to find the intrinsics of the camera, which include the focal lengths (f_x, f_y) and optical centers (c_x, c_y). They are represented in the camera matrix.

$$camera\ matrix: \quad C = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

The camera matrix is very important as we can compute the Projection matrix P using it. If x is a 3D point (real world) in heterogeneous co-ordinates and y be a representation of the image of this point in the pinhole camera, then the following holds.

$$y \sim Px \tag{3.6}$$

Now we can normalize y in the image plane and compute the actual pixel co-ordinates corresponding to the point x.

3.3.3 Matlab Camera Calibration

There are many libraries and tools available to estimate to estimate the parameters discussed in previous section. Matlab camera calibrator app is an easy user-friendly way to get the parameters.

First we need to get a good calibration target. In this project we used a 6X8 chessboard shown in Figure 3.5. We need to collect pictures using our camera at different positions and orientations. We have to maintain the same focus in the camera (turn off auto focus). If the focus changes, we have to recalibrate again. Some cameras change the field of view depending on the frame rate, so it's a good idea to recalibrate for different frame rates. Once we acquire all the images, we can load them into the Matlab Camera Calibrator app. We'll first detect the chessboard patterns in the images. The detected pattern is shown in Figure 3.6.

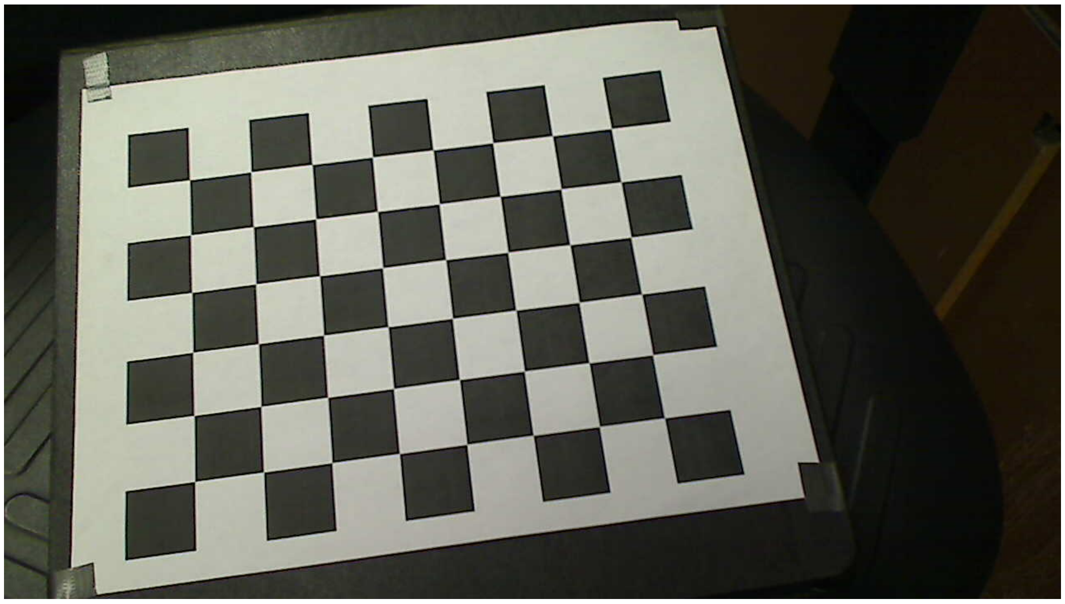


Figure 3.5: Calibration target used in this project

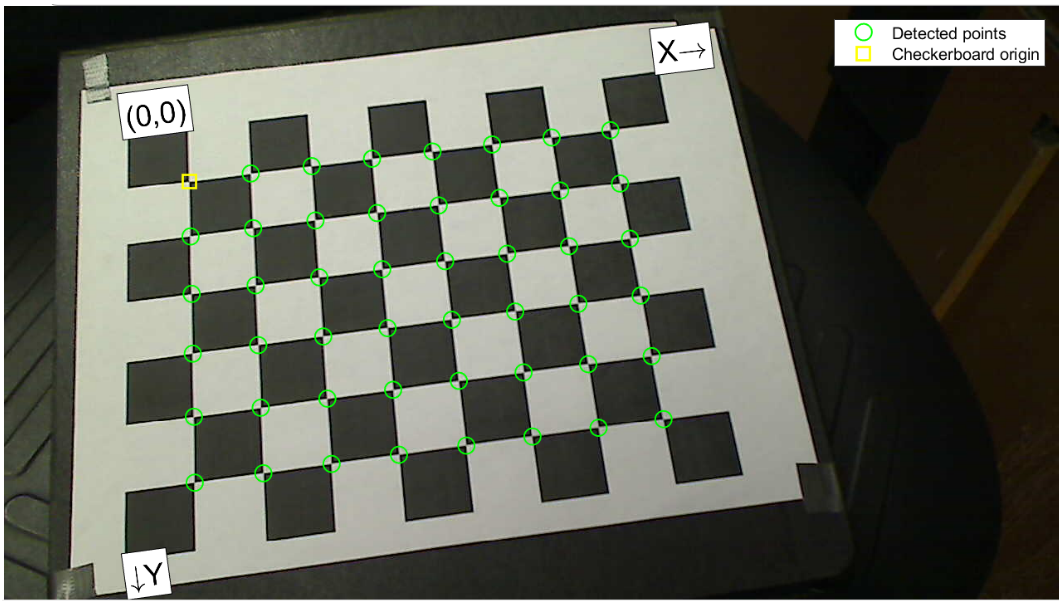


Figure 3.6: Detected the chessboard pattern and identified all the grid points.

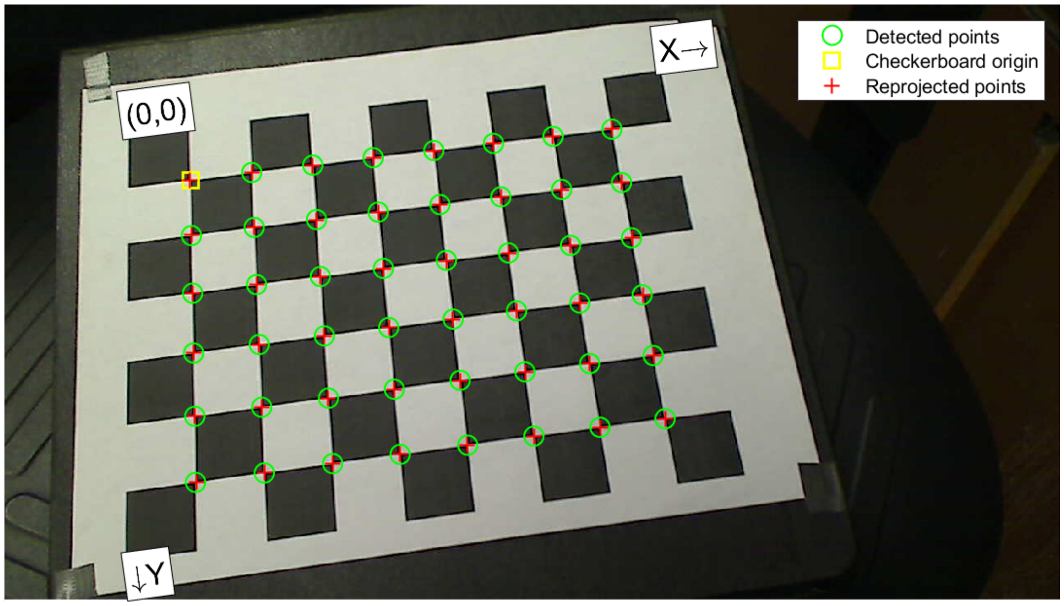


Figure 3.7: Reprojected the grid points (in red) after computing the camera matrix. If the calibration is good, they align perfectly with detected grid points.

We can also see that the origin is set at one of the corners in the board. Now we have two datasets. First, since we know the size of each square in the chessboard (25.4mm), we can compute the real-world co-ordinates of all the grid points identified in Figure 3.6 in the frame of the origin shown in Figure 3.6. Second, We can calculate the co-ordinates of all the grid points in the image (pixel co-ordinates). Using these two datasets and the projection matrix equation shown in section 3.3.2, the app computes the camera parameters. The algorithm used is very similar to OpenCV solvePnP [27].

Once we know the camera parameters, we can re-project the real world calibration target grid points onto the image and see if the points overlap with

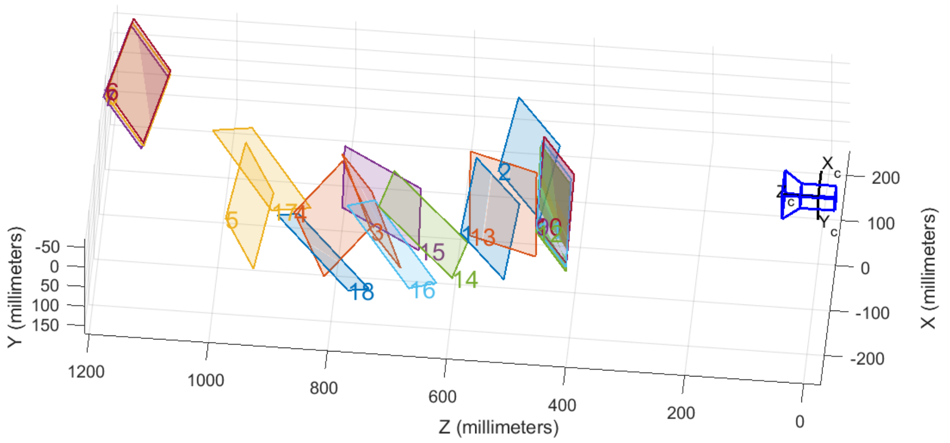


Figure 3.8: Orientations and positions of all the calibration images(shown as colored planes) with respect to the camera (shown in blue at the right end)

the detected grid points, shown in Figure 3.7. For our camera, we observed an average re-projection error of 0.23 pixels. This corresponds to a localization error of 2cm at 2m viewing distance. This calibration is done with 18 images. In Figure 3.8 we can see the orientations and positions of all the images(shown as colored planes) with respect to the camera (shown in blue).

3.4 Apriltags

Apriltags provides us an easy way to localize objects using tags, which are artificial features (visual fiduciaries). Instead of dealing with the cumbersome real-world features we just add a tag to object we need to track. The applications for this method are very diverse as shown in Figure 3.9 Using this we have estimated the camera parameters needed to relate real-world points

Figure 3.9: Different applications for Apriltags. Top left: mobile robot localization. Top right: Boston Dynamics humanoid localization. Bottom left: Tracking ants with Apriltags. Bottom right: VR applcations. Source: [23]

to points on an image. As discussed in section 3.2, we create 2D barcodes as a visual fiduciary tag. This is described in great detail in [20] and [23]. We are using a fast C++ implementation of apriltags algorithm developed by Swatbotics [26].

The pupose of using Apriltags is to localize our robot. We can add a tag to the robot as shown in Figure 3.10. the algorithm first extracts the image points of the 2D bar code, it has two datasets now. First, Since we know the size of each tag, we can compute the real world co-ordinates of the 2D barcode corners in a frame with origin as one of the corners of the code. Two, the

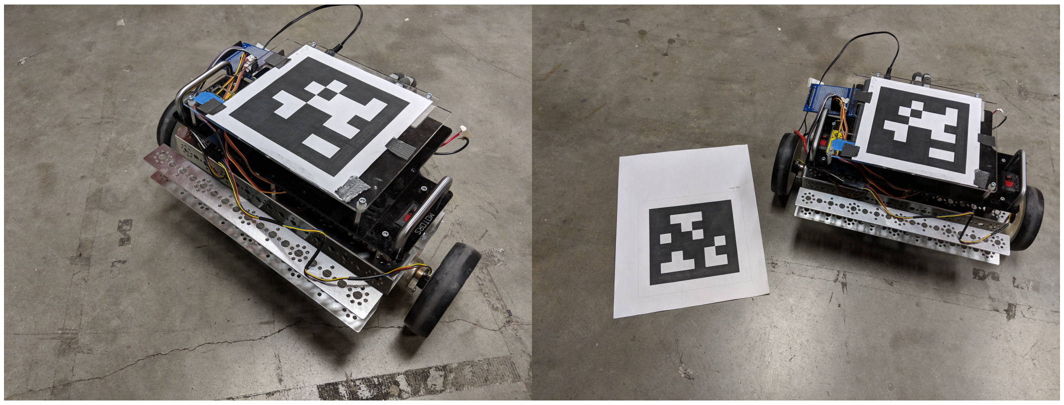


Figure 3.10: The mobile robot Dani with an Apriltag (left). Added ground calibration tag (right)

corner points of the tag on the image. As we saw in section 3.3.3, we can use OpenCV solvePnP algorithm [27] to extract the pose from these two datasets. This process uses the camera matrix data estimated using Matlab in section 3.3.3

Before we start detecting the tags in the image, we have to rectify the image to remove distortions. We can use the camera parameters which we estimated using Matlab. We'll use a ROS node [30] which takes camera parameters and uses equation described in section 3.3.2 to remove the distortion in the image. It gives us a rectified image.

3.4.1 Ground Calibration

Using the OpenCV solvePnP algorithm we get the pose of the tag relative to the camera. The goal of robot localization is to get the position and orientation of the robot in a world frame. The camera is very inconvenient world frame. Since we are working with a 2D robot, it makes sense to have the world frame origin on the ground. We can add another tag to the ground and first get its orientation with respect to the camera (see Figure 3.10). Then we'll make a transformation from the tag frame to the ground frame via the camera frame. This process is made extremely convenient using ROS tf library [31].

3.4.2 Tracking with Apriltags

Now we have a tag on the mobile robot, whose pose we can compute with respect to a ground location (using the ground calibration tag). We can run the detection continuously using a ROS wrapper for apriltags [32]. Experiments showed that we can achieve a detection frequency of 20Hz at a delay of 0.05 seconds. This is suitable for feedback control.

3.5 Localization Trial Data

For a proof of concept, we drove the mobile robot in a perfect circle of radius 0.577m. The position tracking data is shown in Figure 3.11, compared to a perfect circle. After we got the position data, we can differentiate the x,y co-ordinates to get the linear velocity estimate shown in Figure 3.12. We can see that it matches with the velocity commanded. Similarly an estimate for the angular velocity can be obtained by differentiating the orientation data. The control commands given for this trial are

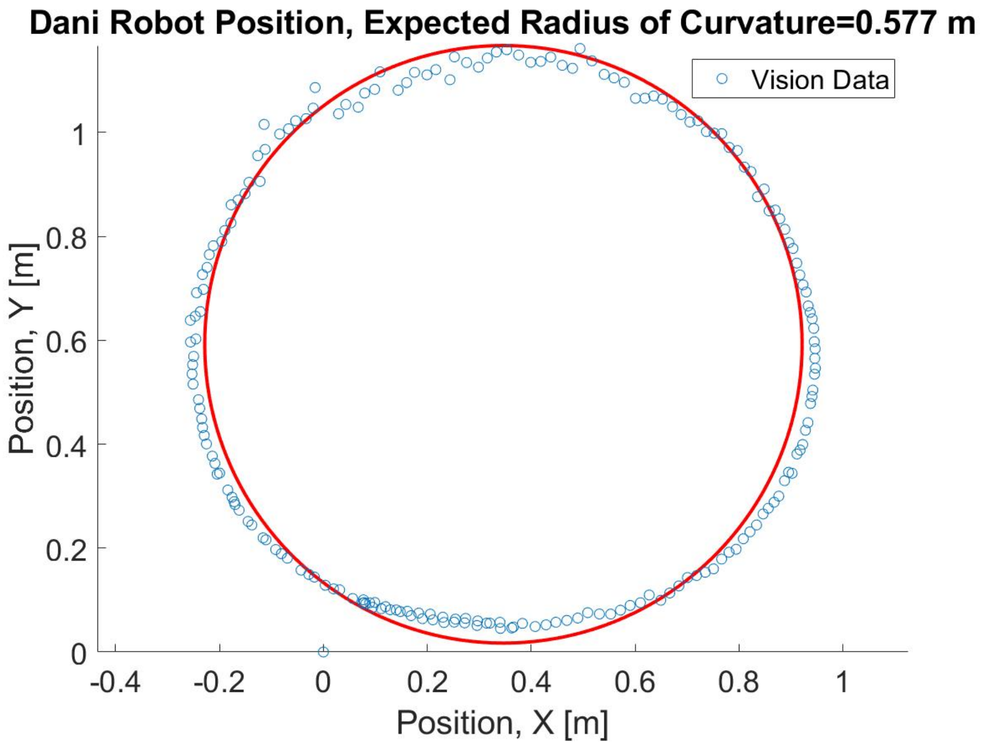


Figure 3.11: Position tracking trial data. The robot is driven in a circle of radius 0.577m.

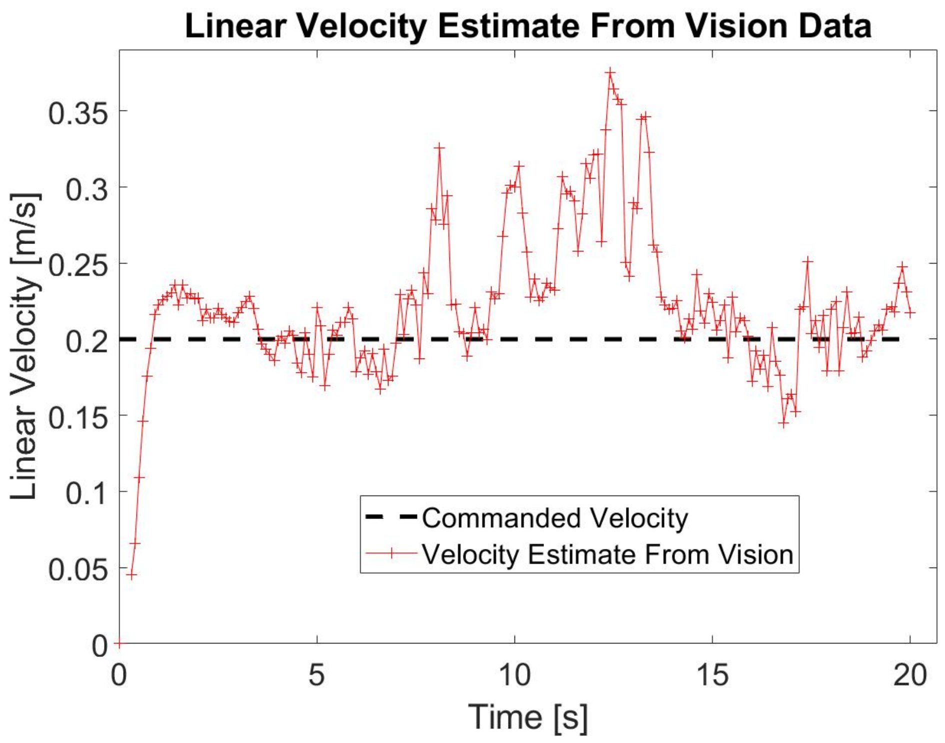


Figure 3.12: Linear velocity data obtained by differentiating the pose data from vision compared to actual velocity.

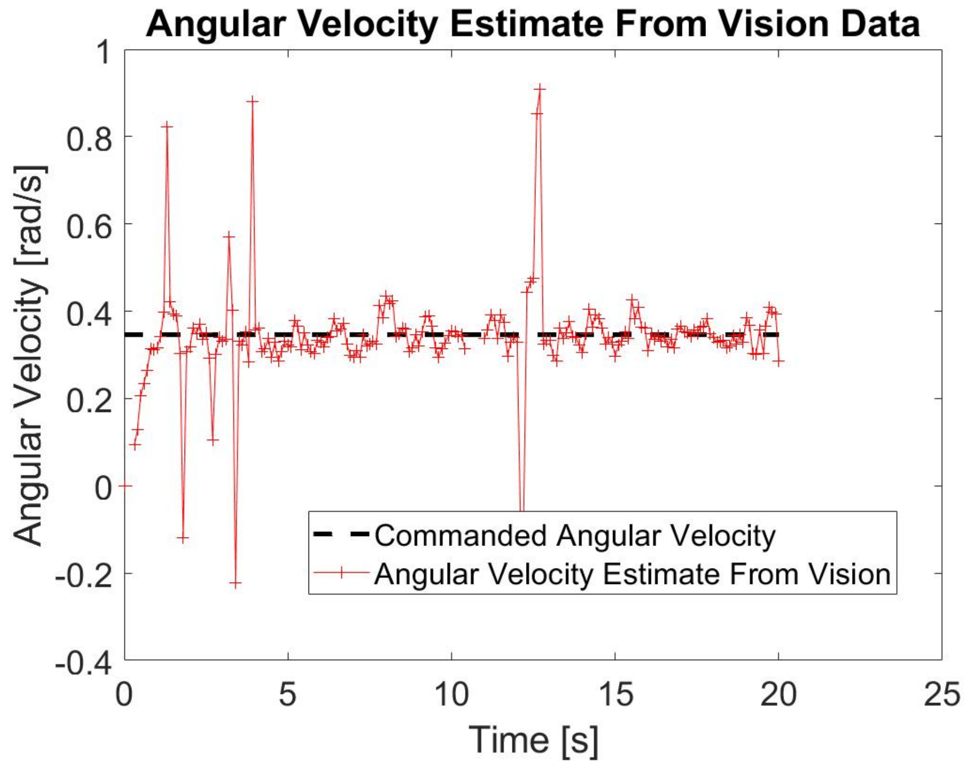


Figure 3.13: Angular velocity data obtained by differentiating the pose data from vision compared to actual angular velocity.

We now have the pose of the robot and the object available in real-time using Apriltags. Next, we analyze the planar pushing problem to compute the primitives for path planning.

Chapter 4

Pushing with a Mobile Robot

4.1 Introduction

In pushing, the grasp planning (to avoid contact loss) and path planning are not independent. We need to continuously ensure that the next steps taken do not lead to contact loss. In most practical cases, we are not aware of the support distribution of the object. This results in the object having multiple solutions for possible motions when pushed with a point contact. If we have multiple contacts (a line contact for example), then we can determine the motion of the object with a unique solution. In Lynch 1992 [18], a *stable push* is defined by pushing an object following a set of the constraints such that there will be no loss of contact with the pushing interface. This means we can consider the object to be rigidly attached to the pushing robot and use constrained path planing for the robot with the object rigidly attached. In [17], the controllability of the stable push is analyzed and a procedure for path planning is described.

4.2 Main Assumptions

The same assumptions made by Lynch and Mason [17] were adopted in this study.

- We are assuming that the Coulomb's Law holds for friction. At a sticking contact, the magnitude of the friction force is less than or equal to μf_n where μ is the static coefficient of friction.
- All pushing forces are in the plane of motion and gravity acts perpendicular to this plane.
- Friction properties are uniform over the plane of pushing
- Process is quasi-static and inertial forces are negligible. The pushing forces are balanced by the frictional force between the object and the ground.

4.3 Pushing Control System

The slider $\mathcal{S}$ (box in our case) is in the world frame $\mathcal{W} = \mathbb{R}^2$ and its configuration space is $\mathcal{C} = \mathbb{R}^2 X S^1$. The slider is pushed by pusher $\mathcal{P}$(mobile robot in our case) at set of points on the perimeter of $\mathcal{S}$. The world frame $\mathcal{F}_{\mathcal{W}}$ (origin $\mathcal{O}_{\mathcal{W}}$) is fixed in the ground plane and the slider frame $\mathcal{F}_{\mathcal{S}}$ (origin $\mathcal{O}_{\mathcal{S}}$) is fixed to the center of $\mathcal{S}$ as shown in Figure 4.1.

Configurations measured in $\mathcal{F}_{\mathcal{S}}$ have the co-ordinates $(x, y, \theta)^T$ and configuration $q = (x_w, y_w, \theta_w)^T$ represents the pose of the slider in $\mathcal{F}_{\mathcal{W}}$. Wrenches

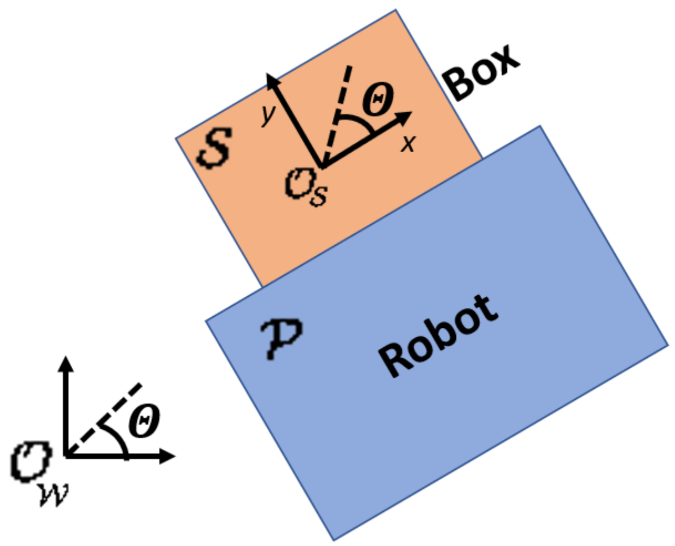


Figure 4.1: The world frame and the slider frame. Source: [17]

f acting on the slider, represented by (f_x, f_y, m) and twist v of the slider are defined in $\mathcal{F}_{\mathcal{S}}$. We are assuming the robot has just enough traction/force capacity to push the box and that all the input energy is lost to friction. For this reason we only need to consider the directions of forces and velocities and not magnitudes. We are making the assumption of a quasi-static system and not considering the dynamics of the system. The force direction $\hat{\mathrm{f}}$ can be represented on a point on the unit circle ($\hat{\mathrm{f}} \in S^2$). Similarly, velocity direction can also be represented by $\hat{\mathrm{v}} \in S^2$. We can represent the velocity direction using a Center of Rotation (COR) in $\mathcal{F}_{\mathcal{S}}$. In figure 4.2, we can see that velocity directions can be directly mapped to a point onto the plane of $\mathcal{F}_{\mathcal{S}}$ about which the velocity direction $\hat{\mathrm{v}}$ is a pure rotation. This representation will be useful in

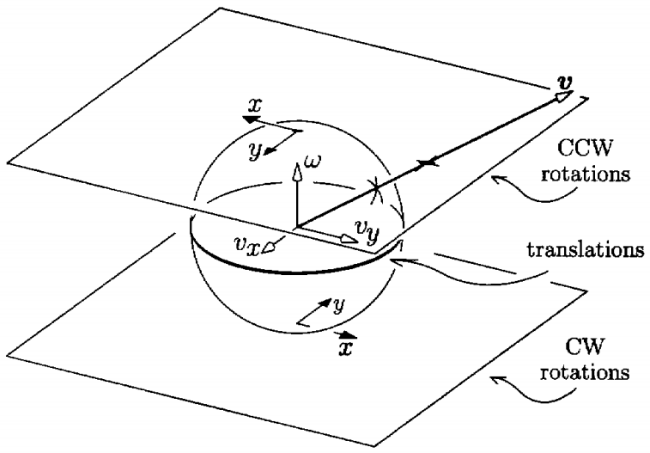


Figure 4.2: The mapping from velocity directions on the unit sphere to rotation centers in the slider frame $\mathcal{F}_{\mathcal{S}}$. Source: [17]

further analysis. The set of force directions that can be applied onto the slider are limited. For example, we can push the slider, but cannot pull it. This leads to non-holonomic constraints on the velocity directions, which means we can't integrate the velocity constraints to get constraints on configuration. The non-linear control system Σ is represented as:

$$\Sigma : \dot{q} = F(q, c_u) = X_u(q)$$
$$q \in \mathcal{C} = \mathbb{R}^2 X S^1, u \in (0, ...n)$$
$$X_u = \begin{cases} 0 & \text{if } u = 0 \\ \begin{pmatrix} cos\theta_w & -sin\theta_w & 0 \\ sin\theta_w & cos\theta_w & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \hat{\text{v}}_{ux} \\ \hat{\text{v}}_{uy} \\ \hat{\omega}_{ux} \end{pmatrix} & \text{otherwise.} \end{cases} \tag{4.1}$$

Here u us chosen from n possibilities of contact configurations (which side of the object do we choose to push on?) and pushing velocities (in which direction are we pushing the slider?) in $\mathcal{F}_{\mathcal{S}}$. For each control input c_u we have a vector field X_u in $\mathcal{F}_\mathcal{W}$. $\hat{\text{v}}_u$ is the velocity direction of slider in $\mathcal{F}_\mathcal{S}$ corresponding to velocity of the slider $X_u(q)$ in $\mathcal{F}_\mathcal{W}$. $\mathcal{X}$ is the set of non-zero vector fields X_u and $\hat{\mathcal{V}}$ is the set of velocity directions $\hat{\text{v}}_u$ in the slider frame $\mathcal{F}_\mathcal{S}$.

Since we have a quasi-static assumption, the path generated in chapter 5 should be considered as a pushing path and not a trajectory. If this is implemented on a robot, the velocities should be small to not deviate far from the quasi-static assumption.

4.3.1 Controllability of the Pushing Control System

To prove the controllability of the pushing control system Σ, we again follow the work of [17]. If the possibilities for velocity directions in $\mathcal{F}_\mathcal{S}$, c_u is $n = 2$, then the Lie algebra $L(\mathcal{X})$ is spanned by X_1, X_2, and $X_3 = [X_1, X_2]$. Here [X,Y] is the lie bracket operation on vector fields X,Y, thus,

$$X_1 = (\hat{\text{v}}_{1x} cos\theta_w - \hat{\text{v}}_{1y} sin\theta_w, \hat{\text{v}}_{1x} sin\theta_w + \hat{\text{v}}_{1y} cos\theta_w, \hat{\omega}_1)^T$$

$$X_2 = (\hat{\text{v}}_{2x} cos\theta_w - \hat{\text{v}}_{2y} sin\theta_w, \hat{\text{v}}_{2x} sin\theta_w + \hat{\text{v}}_{2y} cos\theta_w, \hat{\omega}_2)^T$$

$$X_3 = [X_1, X_2] = (\hat{\omega}_1(-\hat{\text{v}}_{2x} sin\theta_w - \hat{\text{v}}_{2y} cos\theta_w) + \hat{\omega}_2(\hat{\text{v}}_{1x} sin\theta_w + \hat{\text{v}}_{1y} cos\theta_w),$$
$$\hat{\omega}_1(\hat{\text{v}}_{2x} cos\theta_w - \hat{\text{v}}_{2y} sin\theta_w) + \hat{\omega}_2(\hat{-}\text{v}_{1x} cos\theta_w + \hat{\text{v}}_{1y} sin\theta_w), 0)^T$$

The dimension of $L(\mathcal{X}) = rank(X_1\ X_2\ X_3)$. If the determinant of the matrix is non-zero, then the rank is 3.

$$det(X_1\ X_2\ X_3) = (\hat{\omega}_2 (\hat{v})_{1x} - \hat{\omega}_1 (\hat{v})_{2x})^2 + (\hat{\omega}_2 (\hat{v})_{1y} - \hat{\omega}_1 (\hat{v})_{2y})^2$$

The determinant is zero only if: (1) $\hat{\omega}_1 = \hat{\omega}_2 = 0$, whereby the object cannot rotate, or (2) $\hat{\text{v}}_1$ and $\hat{\text{v}}_2$ are multiples of each other, which means the object

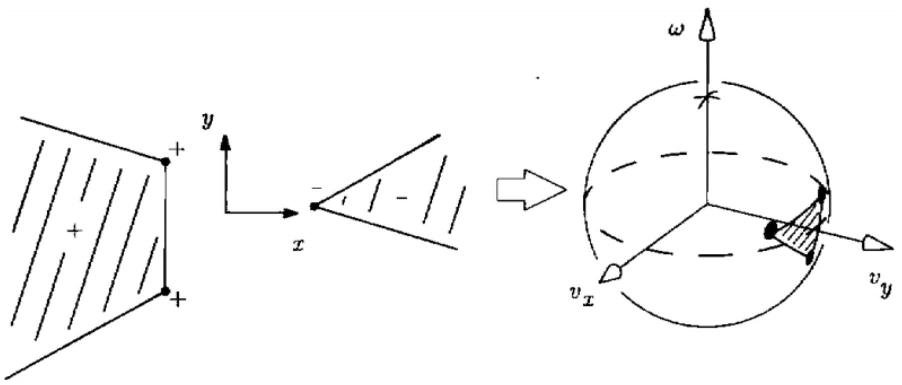


Figure 4.3: The set of velocity directions which can span a great circle represented in rotation center space and velocity sphere. Source: [17]

can move only in one direction. Otherwise, the determinant is non-zero. Now, we have created a new control action using the Lie bracket operation. Since the rank of the Lie algebra is the same as the configuration space, the system is small-time accessible. For Σ, small-time accessibility implies controllability (see [17]).

It can be shown that, the system Σ is small-time locally controllable if and only if the set of velocity directions spans a great circle of the velocity sphere that does not lie in the $\omega = 0$ plane. One example of such a velocity direction set is shown in Figure 4.3.

4.4 Mechanics of Pushing

The force f applied by the pusher is balanced by friction (due to quasi-static assumption). We'll use the following definitions (from [19]):

$$
\begin{aligned}
x &= \text{point of contact } (x, y, 0)^T \text{ between the slider and the support plane} \\
dx &= \text{differential element of support area} \\
p(x) &= \text{support pressure at x} \\
\mu_s(x) &= \text{support friction coefficient at x} \\
s(x) &= \mu_s(x) * p(x) = \text{support friction distribution} \\
v(x) &= \text{linear velocity of x, given by } (v_x - \omega y, v_y - \omega x, 0)^T \\
f_{xy} &= \text{linear components } (f_x, f_y, 0)^T \text{ of f}
\end{aligned}
$$

The origin of the slider frame $\mathcal{O}_{\mathcal{S}}$ is set at the center of friction ($\int_{\mathcal{S}} xs(x)dx = 0$). The force and moment are given by:

$$
\begin{aligned}
f_{xy} &= \int_{\mathcal{S}} \frac{v(x)}{|v(x)|} s(x)dx \\
m\hat{k} &= \int_{\mathcal{S}} x \times \frac{v(x)}{|v(x)|} s(x)dx
\end{aligned}
$$

The friction force always acts in the direction opposite of relative motion. The velocity directions move along the unit sphere shown in figure 4.2, the force f moves along a 2D surface called limit surface (see [6]). At each instant, it can be shown that the velocity direction $\hat{v}$ is normal to limit surface at force f (shown in figure 4.4). If $s(x)$ is finite, the limit surface is smooth and strictly convex.

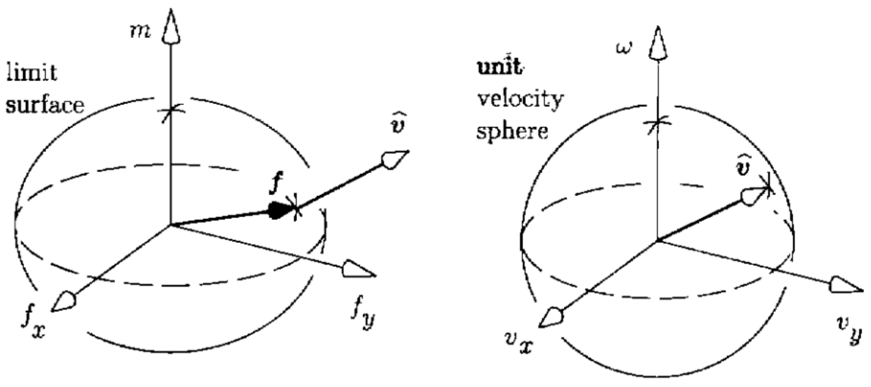


Figure 4.4: Mapping force f to a velocity direction $\hat{v}$, which is normal to limit surface. Source: [17]

The motion of the slider depends on the nature of contact between the pusher and the slider. The contact could be sticking or breaking free or sliding left or right. For each contact mode i, there exists a set of velocities $\mathcal{V}_{k,i}$ that satisfy the contact mode constraints (see [18]). For each contact mode, we can specify composite friction cone containing the feasible forces that can be applied by the pusher (obeying the coulomb friction law) shown in Figure 4.5 a,b. The composite friction cone is mapped onto the limit surface as shown in figure 4.5 c. we know that when the slider is sliding on the ground the velocity direction is normal to the limit surface (as shown in Figure 4.4). We get a set of velocities normal to the composite friction cone on the limit surface (shown in Figure 4.5 c). This set of velocities is $\mathcal{V}_{f,i}$, which follows the force constraints.

The velocity directions in the set $\mathcal{V}_{k,i} \cap \mathcal{V}_{f,i}$ are the feasible directions which follow both kinematic and force constraints. If $\mathcal{V}_{k,i} \cap \mathcal{V}_{f,i} = \phi$, then the

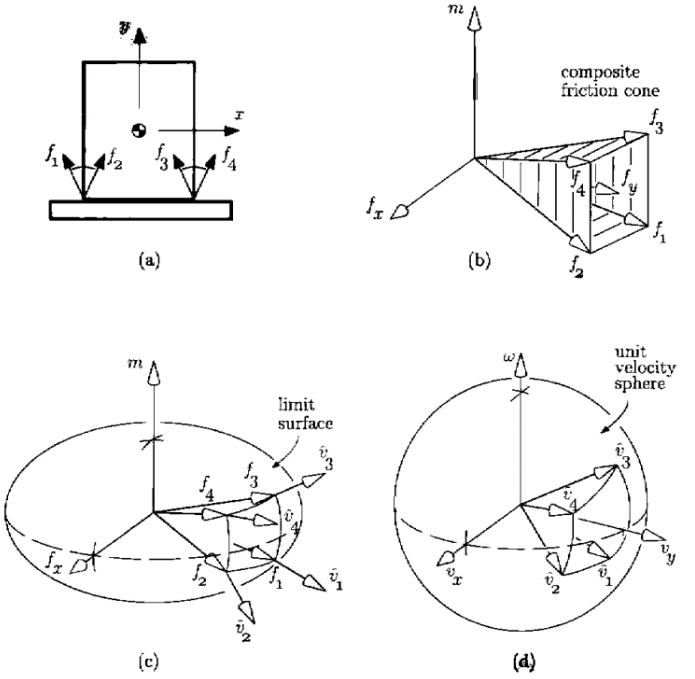


Figure 4.5: (a) Limiting forces that can be applied by the pusher. (b) Convex hull of the limiting forces, called composite friction cone. (c) Composite friction cone on the limit surface and the limiting velocity directions (d) Limiting velocity directions on the unit velocity sphere. Source: [17]

contact mode i is not feasible.

4.4.1 Pushing with Point Contact

When we push the slider with a point contact, three modes are possible: sticking, left sliding and right sliding. The actual motion is given by the contact mode that is consistent with the kinematic and force constraints. The support distribution $s(x)$ is generally unknown. Weaker models for support distributions have been used to study the sliding motion (see [21]). Without, the knowledge of accurate $s(x)$, it is impossible to find exact motion of the slider under point contact pushing. Nevertheless, it has been shown in [17] that a 2-DOF robot (point moving in a plane) can push any object (except a friction less disk) to follow any planar path arbitrarily closely.

4.4.2 Stable Pushing With a Line Contact

Although the controllability for pushing with a point contact has been demonstrated in [17], the motion is very unpredictable because the support distribution is unknown (floor is not flat). So we cannot synthesize controllers for point pushing. If we have two pushing contacts, there will be set of velocities which guarantee sticking condition at all contacts. This is called a *stable push*. We will use the *stable push* directions to get the primitives for path planning.

Let $\hat{\mathcal{V}}_{stable}$ be the set of velocity directions which result in a *stable push*. Let $\hat{\mathcal{V}}_{\mathcal{F}}$ be the set of directions such that one solution of the motion of the slider is to remain fixed to the slider.This set of velocity directions is found by intersecting the composite friction cone $\mathcal{F}$ from the line contact with the limit surface as shown in Figure 4.5 c. In [18], a method is proposed to identify a *subset* (or an approximation) of $\hat{\mathcal{V}}_{\mathcal{F}}$ for a slider with a known center of friction. It can be shown that this set also belongs to $\hat{\mathcal{V}}_{stable}$.

We follow a procedure STABLE as described in [17] to identify a subset of $\hat{\mathcal{V}}_{\mathcal{F}}$. This is a refinement of the procedure shown in [18]. In STABLE we identify regions surrounding the object, in which we can place a center of rotation for the object without losing contact with the pushing interface.

Procedure STABLE:

1. We have two friction cones at the edges of the contact interface as shown in Figure 4.6a (green arrows) with angle $tan^{-1}\mu$ with the contact normal. For each edge of the two friction cones, draw lines perpendicular to the edge, such that the entire object is contained within the 4 lines. We'll have a region on each side of the object. See Figure 4.6a These regions contain the rotation centers which can be achieved by applying forces within the angle limits of the friction cone.

2. We have two end points for the contact interface (line). For each end point, draw two lines (yellow) perpendicular to the line (red) joining the end point and the center of friction. One line is a perpendicular bisector of the line joining the end point and center of friction and the other line is at a distance of r^2/p on the other side of the center of friction. r is the distance from center of friction to the most distant support point on the object and p is the distance from end point to the center of friction. This step will also give us two regions on either side of the object, which contains the possible centers of rotation for forces that pass between the two end points. This region is shown in figure 4.6b.

3. The intersection of the regions found in steps (1) and (2) gives us a region of possible centers of rotation, which guarantees no loss of contact with

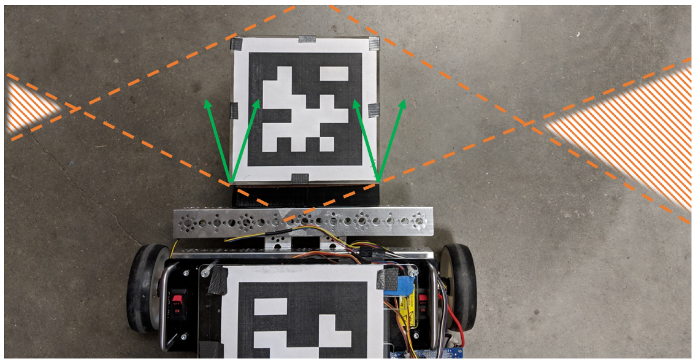
the pushing interface in a quasi-static process. This region is shown in Figure 4.6c.

The procedure STABLE is not exhaustive, it misses some points with low radius of curvatures. We have a conservative estimate of the possible center of rotations for the object using a line contact for pushing. Theorem 4 in [17] proves that the subset of $\hat{\mathcal{V}}_{\mathcal{F}}$ identified in STABLE is also a subset of $\hat{\mathcal{V}}_{stable}$. Controllability analysis for pushing with a line contact using the velocity directions found in procedure STABLE is shown in [17]. In order to move an object from an initial pose to a final pose, at each point we need to perform the procedure STABLE, identify feasible centers of rotation and use the primitives for path planning. The path planning procedure is discussed in Chapter 5.

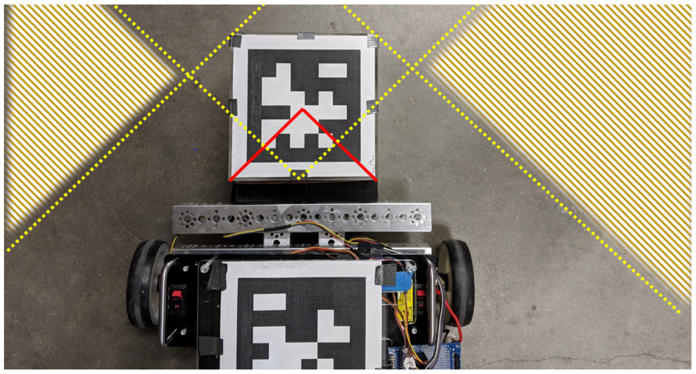
4.5 Pushing with a Mobile Robot

Pushing with a mobile robot simplifies the planar pushing problem, by constraining the region of feasible center of rotations (CoRs) found in procedure STABLE. This will reduce the maneuverability compared to a planar manipulation by a pusher/bumper attached to robotic arm. Since we are using a differential drive robot, the possible centers of rotation lie on a line passing through the center of the driving wheels as shown in figure 4.7. We need to find the intersection of the line (of possible centers of rotation for the differential drive robot) and the region found in procedure STABLE.

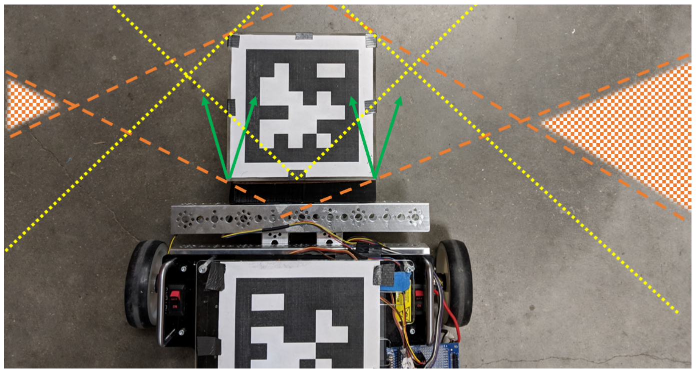
In order to push and object with a mobile robot, we need to select



(a)



(b)



(c)

Figure 4.6: Procedure STABLE. The shaded region in 4.6c is where we can select a center of rotation for the robot for stable pushing.

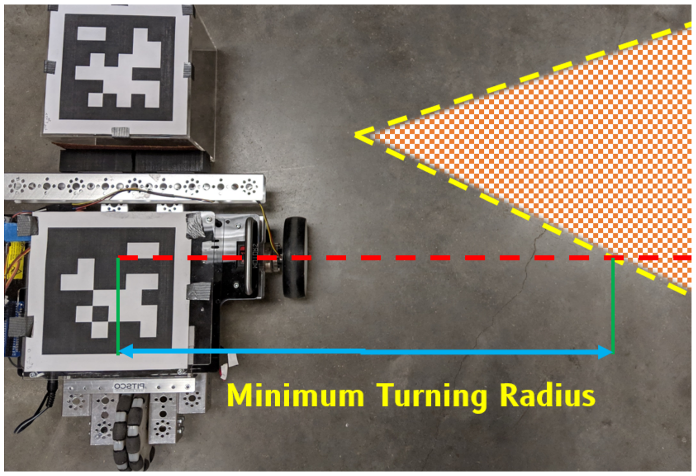


Figure 4.7: Computing the minimum turning radius which guarantees stable pushing with a mobile robot.

points on the section of the straight line identified in Figure 4.7. **This is interpreted as having a minimum turning radius for the robot.** For the dimensions of the robot and the box used in this study, the minimum radius depends only on the coefficient of friction μ between the box and the robot's pushing surface. The trend is shown in Figure 4.8. The only dimension that affects this trend line is the distance between the box center and the robot's center. The coefficient of friction μ is measured experimentally to be in the range of 1.5-2. From Figure 4.8, we can see that the minimum turning radius predicted is $\sim$0.35m. This claim is tested in Section 6.1.

Now we have the minimum turning radius feasible for a given contact configuration. Using this information, we will plan paths around obstacles in the next chapter.

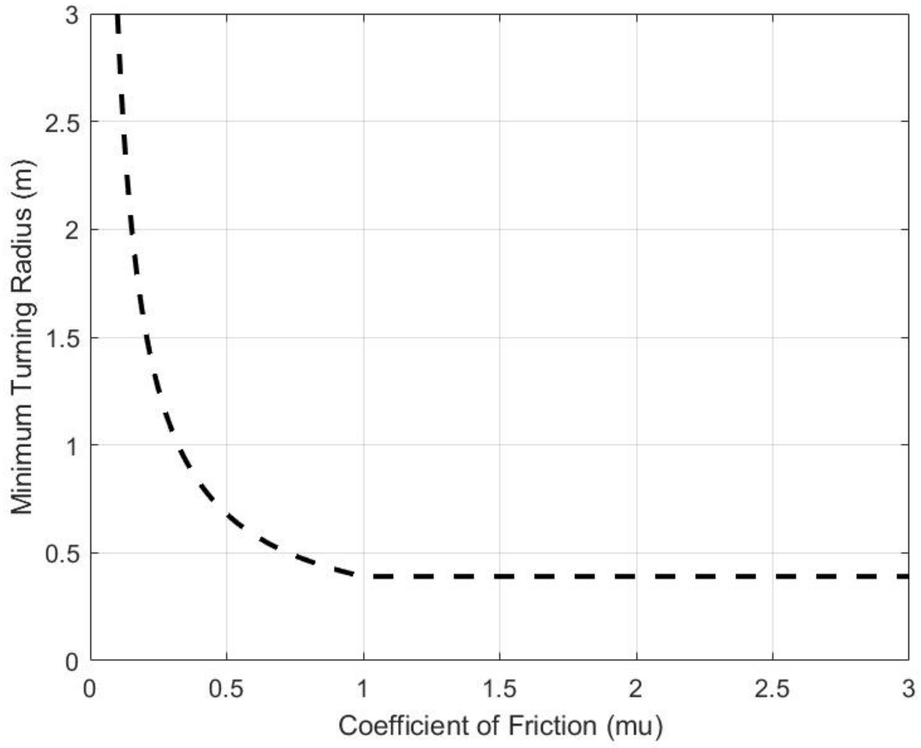


Figure 4.8: Minimum turning radius for the robot vs Coefficient of friction between the box surface and the robot's pushing surface.

Chapter 5

Path Planning for Controlled Box Pushing

5.1 Introduction

As seen in Chapter 2, the robot is controlled through velocity, V, and yaw velocity, ω, commands, with an admissible velocity space $\mathbb{R}^2$. The state vector of the robot is $[x, y, \theta]$, with configuration space $C = \mathbb{R}^2 \times S^1$. The dimension of configuration space (3) is greater than admissible velocity space (2). This means that we have a non-holonomic system. There are many algorithms available in the literature for path planning for a non-holonomic system [12],[13]. In this study we considered two algorithms, A* search and Rapidly exploring Random Trees (RRT). We also have a minimum turning radius constraint as discussed in Section 4.5.

5.2 A* Search

A* search was proposed by Hart, Nilsson and Raphael [7]. It is considered an extension to Dijkstra's shortest path search algorithm, but with a performance improvement using heuristics. A detailed treatment can be found in Latombe [12], for planning with non-holonomic constraints using A* search.

For application to this study, the workspace area was divided into cells of size 1 inch X 1 inch (25mmX25mm). The configuration space $C = \mathbb{R}^2 \times S^1$ is discretized with an angle (robot's orientation) resolution of 2 degrees. Now, we have a 3D grid (representing x,y, θ) with a starting node and a goal node corresponding to starting and goal poses of the robot respectively. The goal is to find a feasible path from the starting node to goal node, avoiding all the obstacles. We represent the obstacles in the physical world using an occupancy grid (refer to [3]). A sample map with obstacles, starting and goal poses is shown in 5.2. A* search is guaranteed to produce the shortest path between two nodes if it exists.

We maintain two lists of nodes: OPEN and CLOSED. The list OPEN contains the nodes whose children haven't been explored yet. The list CLOSED contains the nodes whose children are explored. The list OPEN is maintained in a sorted order using the heuristic cost $f(n)$,

$$f(n) = g(n) + h(n), \tag{5.1}$$

where n is the node under consideration. The term $g(n)$ represents the cost of taking the shortest path from the starting node to n, and $h(n)$ is the heuristic cost from the node n to the goal node. A* search can be implemented more efficiently if the heuristic is admissible. It has to satisfy the following constraint:

$$h(x) \leq d(x, y) + h(y), \tag{5.2}$$

where $d(x, y)$ is the length of the edge connecting (x, y). The list OPEN is implemented as a priority queue. In each iteration, we take the node at the top of the list OPEN (it will have the least heuristic cost) and generate a list of nodes the robot can reach subject to the planar pushing constraints. The reachability set of the robot is constructed by computing the robot's motion for all the permissible control actions for a fixed time step. In our case, the radius of curvature for the path of the robot is the control variable. The different locations reached by the robot for different radius's is shown in Figure 5.1.

Now, for each new node reached, we compute the heuristic cost and add it to the list OPEN. If the node was already visited, the heuristic cost is updated if changed. If the node was already visited and is in list CLOSED, the node is ignored. After this step, the node at the top of list OPEN is erased and added to list CLOSED. This process is repeated until the goal node appears at the top of list OPEN.

It is important to note that we are trying to reach a goal neighborhood and not the exact goal (which takes a lot of compute resources). The step size should be such that it is not too big it will skip the goal neighborhood and not too small such that the search tree is too large. We will design the data structure such that, we store the parent node for each of the node visited. Once we finish the iterative process above, we can simply trace back the parents starting from the goal node and reaching the starting node. We will get the shortest path in this process. A sample A* path is shown in figure 5.2 (with safety margins from obstacles).

The A* star search algorithm has a time complexity of $O(b^d)$ (depending on the heuristic cost) where b is the branching factor and d is the depth of

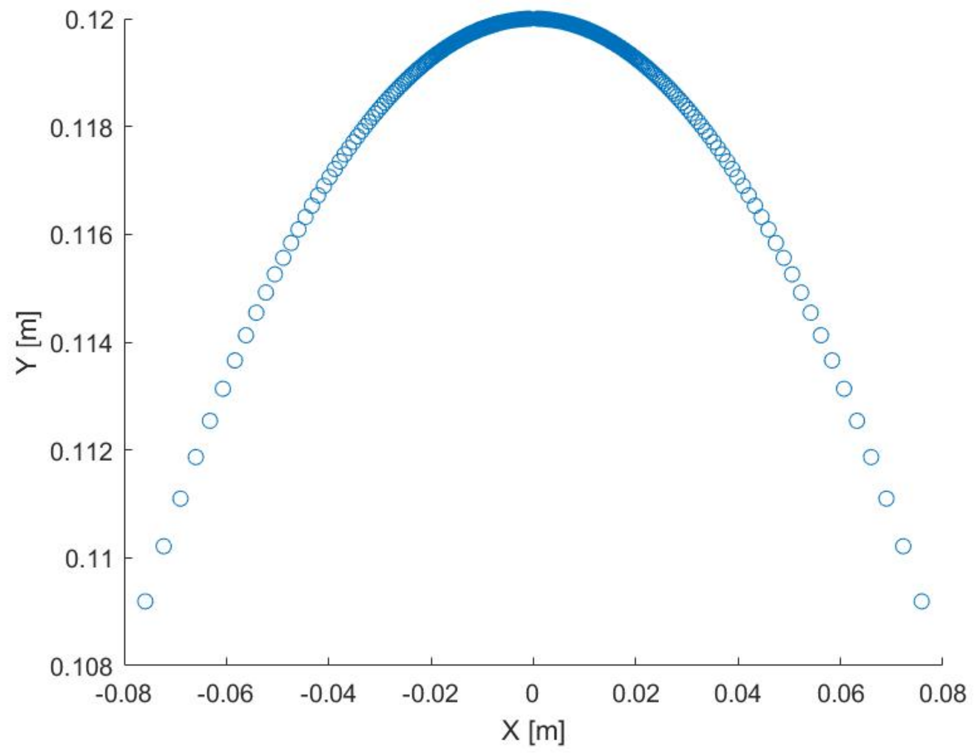


Figure 5.1: The reachability set of the robot (x,y co-ordinates) with the constraint of stable pushing. (Path step size is 0.1m)

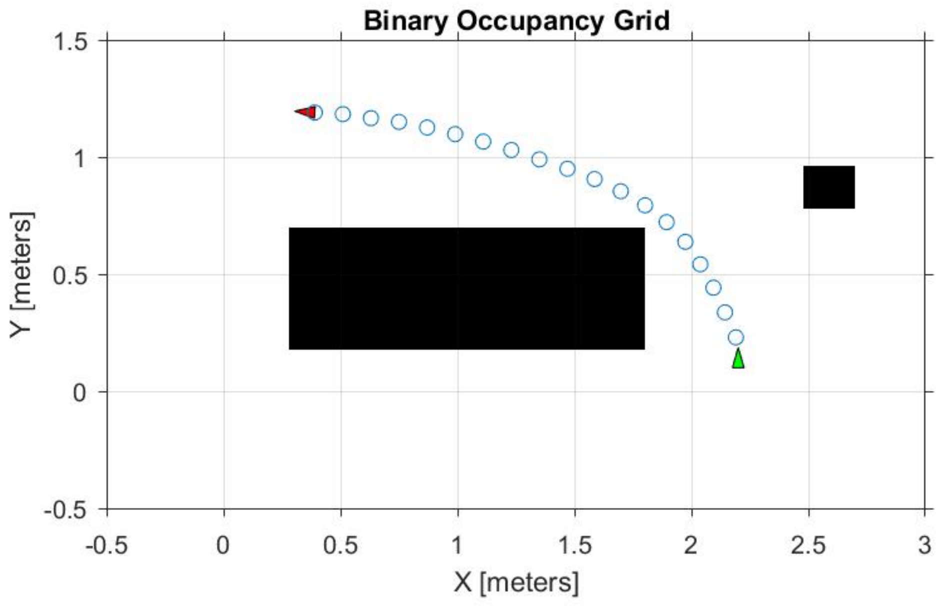


Figure 5.2: A* path from starting pose (green) to a goal pose (red) around obstacles (black).

the solution. The memory requirements are also very large depending on the grid size and dimension of the configuration space. These space and time requirements might make some planning problems computationally expensive and not suitable for real time planning. One alternative is to use ARA* [16]. In the next section, we describe sampling based planning.

5.3 Rapidly Exploring Random Trees

In A* search we explored new nodes deterministically; i.e., we explored the neighbors of the current node. In sampling based planning exploration is probablistic. One such planning algorithm is Rapidly exploring Random Tree (RRT), first proposed by LaValle and Kuffner [14]. Their approach converges to a non-optimal solution with probability of 1. RRT* is an improvement [9] and is globally asymptotically optimal for holonomic systems. Our problem has non-holonomic constraints (e.g., we can't move sideways). The RRT* method was extended to non-holonomic systems by Karaman 2010 [10]. For the implementation steps, we refer the reader to [10]. A sample path generated using RRT to push a box into a C-shaped receptacle using a mobile robot is shown in Figure 5.3.

5.4 Trajectory Tracking

We obtained a desired path from the planning process in the previous sections and we want the robot to follow this path minimum deviation. There are a lot of sophisticated algorithms proposed in the literature, see [8], [2],

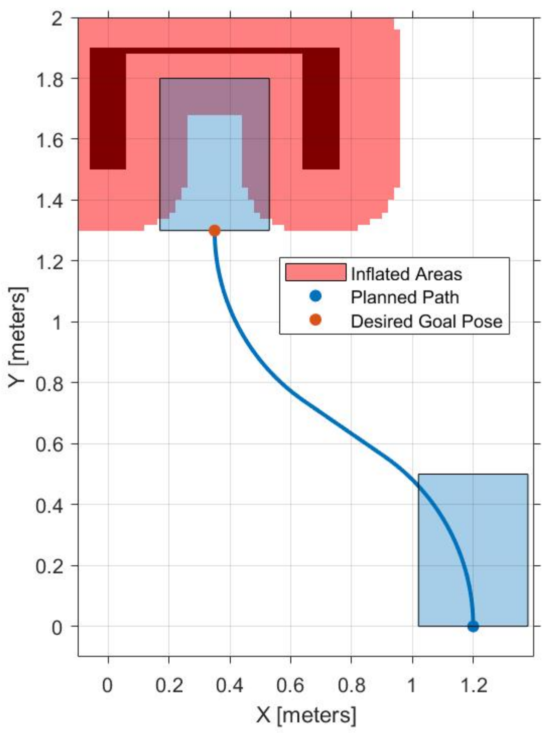


Figure 5.3: Path planned using RRT with a "C" shaped obstacle into which we want to push the box.

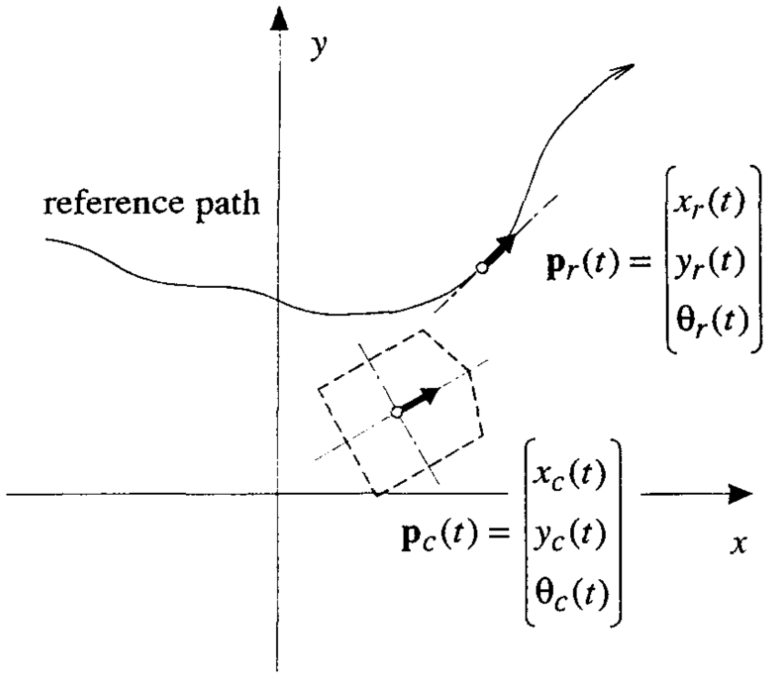

y
reference path
$\mathbf{p}_r(t) = \begin{bmatrix} x_r(t) \\ y_r(t) \\ \theta_r(t) \end{bmatrix}$
$\mathbf{p}_c(t) = \begin{bmatrix} x_c(t) \\ y_c(t) \\ \theta_c(t) \end{bmatrix}$
x

Figure 5.4: Robot's current pose P_c and the reference pose P_r in trajectory tracking. Source: [8]

[11]. The control proposed in [8] is Lyapunov stable. We can also estimate the system parameters b (width of the robot) and R_{wheel}, as shown in [2]. In this study, we are assuming that the measured values for b and R_{wheel} are accurate. We will use a control scheme similar to the approach in [11]. We'll assume that the ideal pose of the robot is moving with a constant velocity along the planned path. We have $V = 0.15m/s$ and ω is computed using the θ trajectory of the planned path. In the figure 5.4, we can see the robot's current pose $P_c(t)$ and the desired pose $P_r(t)$ at time t. It is advantageous to compute the relative pose of P_r w.r.t. P_c; i.e., compute $P_r(t)$ relative to the

robot's frame (X_R, Y_R) centered at P_c, as shown in Figure 2.2. The relative pose becomes the error that we need to minimize for the robot to follow the path.

$$\begin{aligned} x_e &= (x_r - x_c)cos\theta_c + (y_r - y_c)sin\theta_c \\ y_e &= -(x_r - x_c)sin\theta_c + (y_r - y_c)cos\theta_c \\ \theta_e &= \theta_r - \theta_c \end{aligned} \tag{5.3}$$

Now our goal is to have (x_e, y_e) and θ_e as close to 0 as possible. For that purpose we use the following control law:

$$\begin{aligned} V &= k_1 x_e \\ \omega &= k_2 y_e + k_3 \theta_e \end{aligned} \tag{5.4}$$

Here V and ω are the linear velocity and angular velocity inputs for the mobile robot. k_1 and k_2 are positive. It can be demonstrated that the control law results in having $(0, 0)$ as a local attractor for the error dynamics (See appendix of [11]). Experimental data demonstrating the trajectory tracking control law described above is seen in Figures 6.3 and 6.4.

Chapter 6

Pushing Experiments

The approach in [17] is mainly designed for open loop pushing and it is the reason why we want to avoid slip between the slider and pusher. This open loop approach although successfully implemented, has not been tested at the limits (turns with low turning radius). Previous studies used conservative estimates of friction coefficients and ensured a no slip condition between the slider and pusher by having a huge factor of safety. Now, we have computer vision systems which can be easily setup and they are capable of providing high fidelity pose data [23], which can be used to test the system. A computer vision system based on Apriltags was used to get high quality feedback, which was used for trajectory tracking and visualizing pose data in the pushing experiments. An overview of the full system is given in Figure. 6.1.

6.1 Validation of Minimum Turning Radius

We want to validate the minimum turning radius suggested by the analysis in [17], shown in Figure 4.8.We measured the coefficient of friction between the robot's pushing interface and the box to be in the range of 1.5-2.

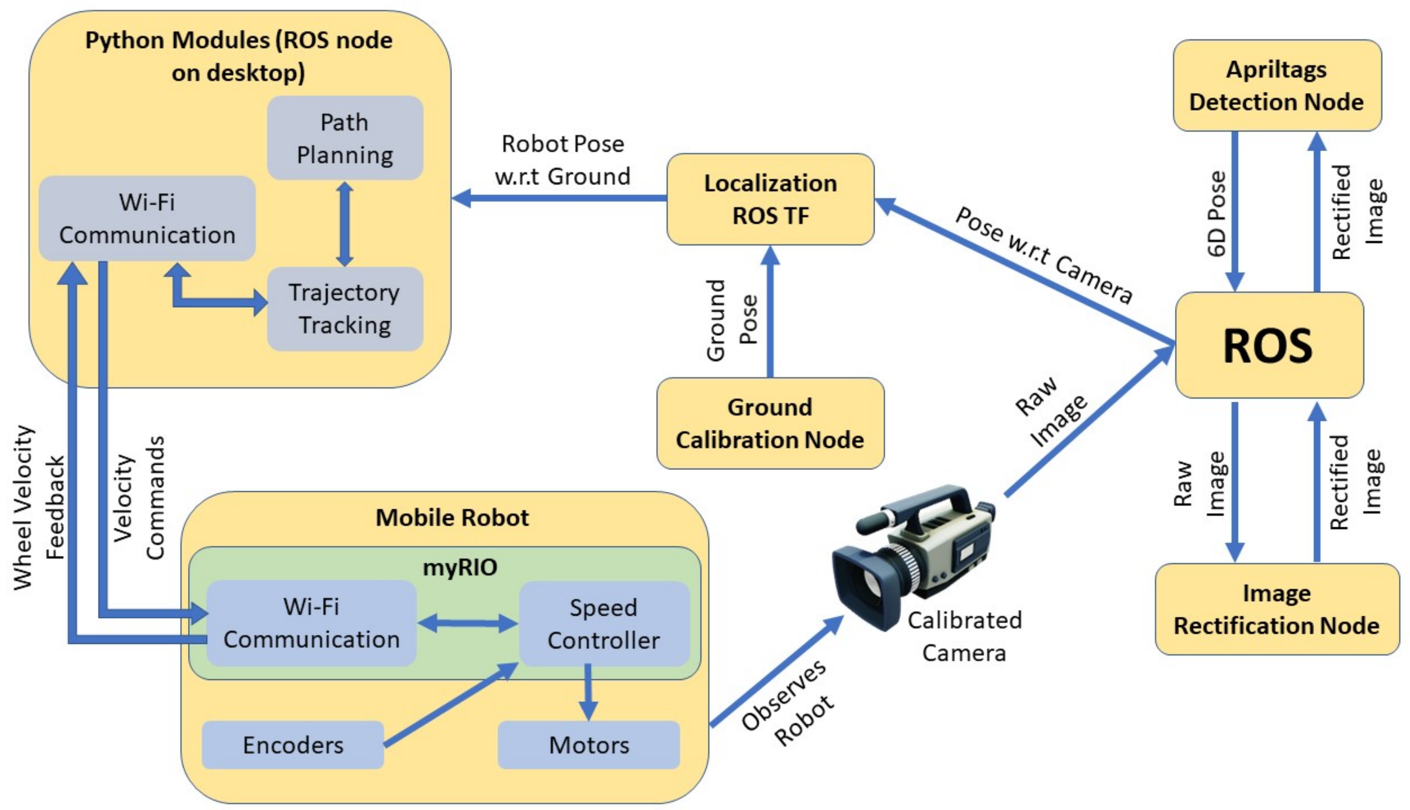


Figure 6.1: Overview of the complete system developed in this study.

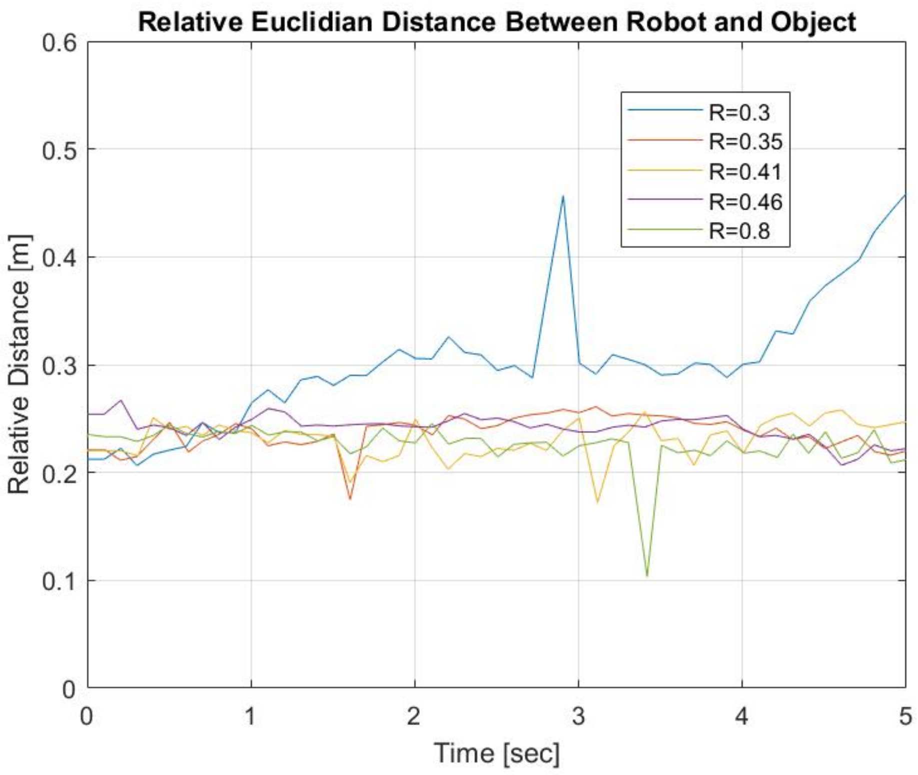


Figure 6.2: Relative distance between the robot center and the object center while pushing in circular path.

From figure 4.8, we can see that the minimum turning radius is 0.38m. This is a conservative estimate (see [17]), which means that if we push the box in a circular path, there will be slip at the pushing interface if the path radius is less than 0.39m. To validate this, we pushed the box in circular paths of varying diameters. In the figure 6.2, we can see that the relative distance remains constant for all the path radii except for R=0.3m (ignoring the noise from vision and ground roughness). This means contact slip starts occurring at some $R < 0.35m$. Since the estimate of 0.39m as minimum turning radius (see

Section 4.4.2) is conservative, the result shown in Figure 4.8 is experimentally valid.

6.2 Box Pushing Task

Once we have validated the minimum turning radius result, we can plan more complex paths for the box pushing task. Once we plan a path, we can use the control law proposed in section 5.4 to follow the path. Since the path planning is done with a minimum turning radius as 0.39m, we can be sure that the object doesn't slip at the pushing interface. Experimental data for a pushing box along a planned path can be seen in figure 6.3.

In Figure 6.3, we can see that although there is initial disturbance, the robot returns to the nominal path since we have a stabilizing control action. This is more clearly illustrated in figure 6.4, the orientation of the box returns to the nominal trajectory after an initial disturbance. This control scheme for pushing is robust and in this study, we are only limited by the spatial constraints imposed by the limited field of vision of the cameras. The data from multiple pushing trial is shown in Figure 6.5 and the errors in trajectory tracking are reported in Table 6.1.

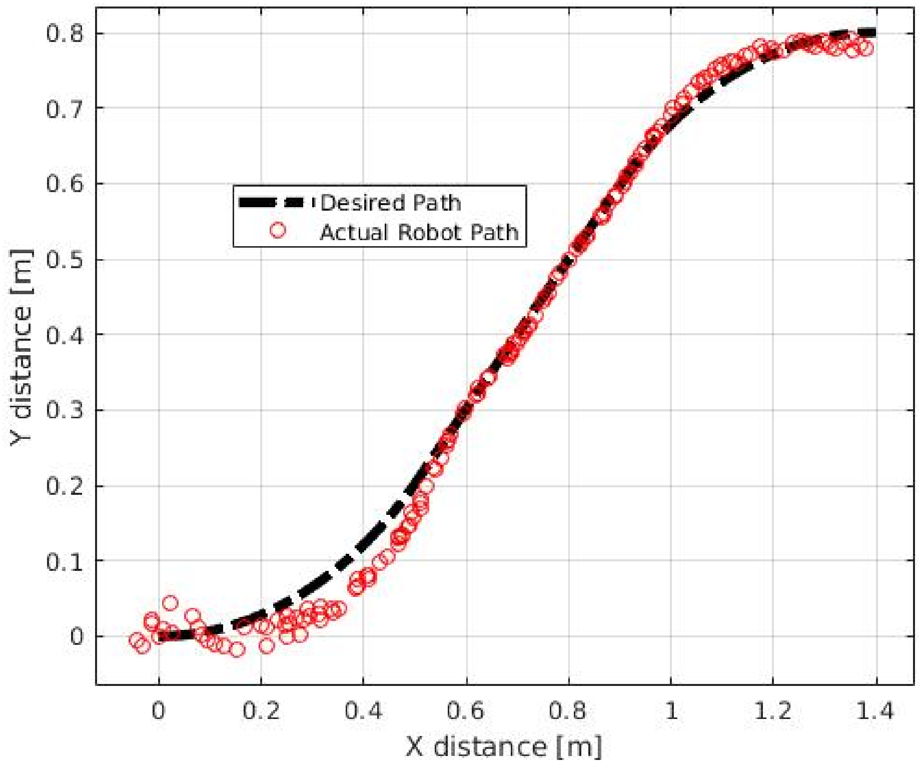


Figure 6.3: Path of the center point of the box while being pushed by the robot along a planned path.

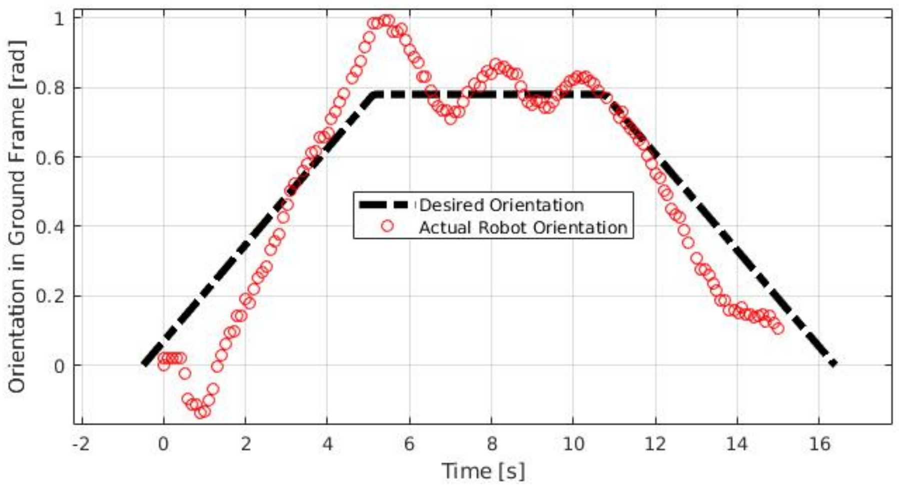


Figure 6.4: Orientation of the box, while being pushed along the planned path in figure 6.3.

Trial	Maximum Spatial Error
1	8.5cm
2	6cm
3	8.4cm
4	7.5cm
5	9.2cm

Table 6.1: Maximum errors in trajectory tracking in different trials.

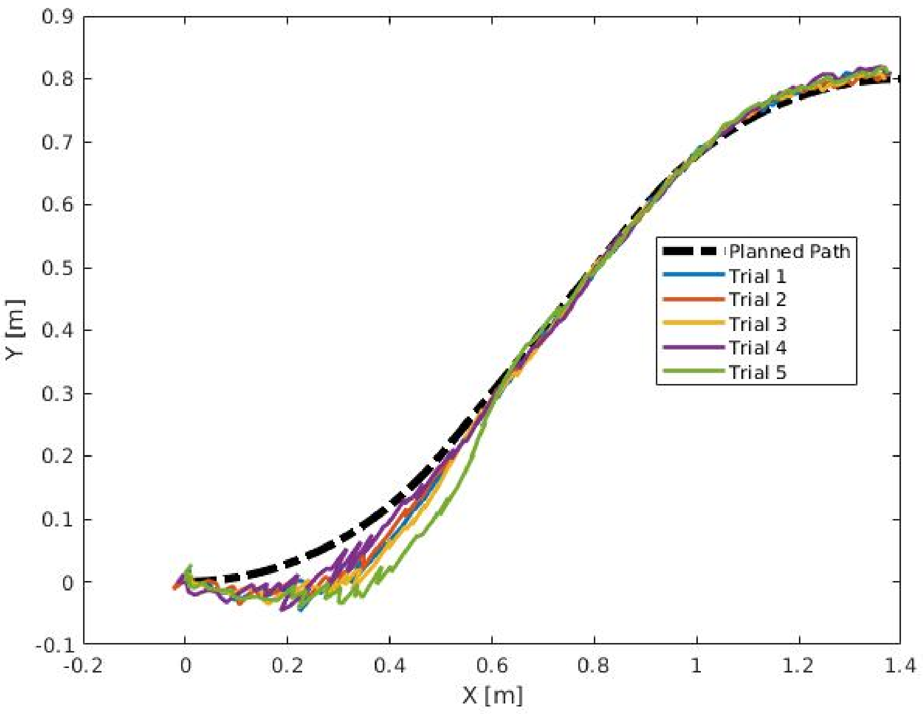


Figure 6.5: Paths of the center point of the box while being pushed by the robot along a planned path during multiple trials.

Chapter 7

Conclusion

In this study, we have successfully validated the stable pushing analysis in Lynch and Mason 1996 [17] by designing and conducting experiments using a small mobile robot platform. We have used the analysis in [17] and implemented path planning and control algorithms that run in real-time. In [17], Dijkstra's algorithm has been used for path planning, while we used A* search and RRT* for planning which are more efficient in memory and time.

This system is able to successfully plan paths for a box pushing task with arbitrary starting and ending poses (as long as a feasible path exists), as demonstrated with results where trajectories of the mobile robot pushing a box through planned paths are shown. Having object pose feedback helped us to use the minimum turning radius possible for the system (frictional limit) for path planning and perform successful trajectory tracking. Previous studies [17] had safety margins for the minimum turning radius (which is a result of a conservative estimate of coefficient of friction) so that they can ensure no slip between robot and object in open loop trajectory tracking schemes.

The control policy in section 5.4 could be improved by applying Model Predictive Control (MPC), which could potentially handle turns better than the simple proportional control implemented in this study. The proportional control is “reactive” in nature; i.e., if there is a turn coming up in the path, the yaw commands are given only after the robot actually enters the turn. An MPC approach would use a look ahead horizon and would give control inputs considering the future trajectory, leading to better trajectory tracking performance.

Pushing with a mobile robot simplified the planar pushing problem by constraining the feasible region of center of rotations. If we are instead pushing with a bumper attached to a robotic arm, the planning becomes more complex. Machine Learning algorithms might be more successful in those problems. Reinforcement Learning algorithms [15] could be applied to learn the box pushing task. Initially we can have the goal position fixed relative to the starting position and develop a controller that handles disturbances better than the control we have demonstrated in Section 5.4. The next step would be to change the goal position relative to the initial position with a static obstacle map.

Bibliography

[1] Maria Bauzá and Alberto Rodriguez. A probabilistic data-driven model for planar pushing. *CoRR*, abs/1704.03033, 2017. URL: `http://arxiv.org/abs/1704.03033`.

[2] Wenjie Dong and K-D Kuhnert. Robust adaptive control of nonholonomic mobile robot with parameter and nonparameter uncertainties. *IEEE Transactions on Robotics*, 21(2):261–266, 2005.

[3] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[4] Levine et al. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. RSS, 2018.

[5] OpenAI et al. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018. URL: `http://arxiv.org/abs/1808.00177`.

[6] Suresh Goyal, Andy Ruina, and Jim Papadopoulos. Planar sliding with dry friction part 1. limit surface and moment function. *Wear*, 1991.

[7] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[8] Yutaka Kanayama, Yoshihiko Kimura, Fumio Miyazaki, and Tetsuo Noguchi. A stable tracking control method for an autonomous mobile robot. In *Proceedings., IEEE International Conference on Robotics and Automation*, pages 384–389. IEEE, 1990.

[9] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *arXiv preprint arXiv:1005.0416*, 2010.

[10] Sertac Karaman and Emilio Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *49th IEEE conference on decision and control (CDC)*, pages 7681–7687. IEEE, 2010.

[11] Kyoung Chul Koh and Hyung Suck Cho. A smooth path tracking algorithm for wheeled mobile robots with dynamic constraints. *Journal of Intelligent and Robotic Systems*, 24(4):367–385, 1999.

[12] Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

[13] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[14] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

[15] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[16] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. Ara: formal analysis. 2003.

[17] Kevin Lynch and Matthew T. Mason. Stable pushing: Mechanics, controllability, and planning. *International Journal of Robotics Research*, 1996.

[18] Kevin M Lynch. The mechanics of fine manipulation by pushing. *ICRA*, pages 2269–2276, 1992.

[19] Matthew T Mason and J Kenneth Salisbury Jr. Robot hands and the mechanics of manipulation. 1985.

[20] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. ICRA, 2011.

[21] Michael A Peshkin and Arthur C Sanderson. The motion of a pushed, sliding workpiece. *IEEE Journal on Robotics and Automation*, 4(6):569–598, 1988.

[22] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots.* Bradford Company, Scituate, MA, USA, 2004.

[23] John Wang and Edwin Olson. Apriltag 2: Efficient and robust fiducial detection. IROS, 2016.

[24] Kuan-Ting Yu, Maria Bauzá, Nima Fazeli, and Alberto Rodriguez. More than a million ways to be pushed: A high-fidelity experimental data set of planar pushing. *CoRR*, abs/1604.04038, 2016. URL: `http://arxiv.org/abs/1604.04038`.

[25] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

[26] Fast c++ implementation of apriltags. URL: `https://github.com/swatbotics/apriltags-cpp`.

[27] Opencv solvepnp algorithm to compute pose. URL: `https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`.

[28] Radial distortion correction. URL: `http://www.uni-koeln.de/~al001/radcor_files/hs100.htm`.

[29] Robot navigation in wikipedia. URL: `https://en.wikipedia.org/wiki/Robot_navigation`.

[30] Ros image processing node. URL: `http://wiki.ros.org/image_proc`.

[31] Ros transformation library (ros tf). URL: `http://wiki.ros.org/tf`.

[32] Ros wrapper for the apriltags library. URL: `https://github.com/personalrobotics/apriltags`.

[33] Sward camera calibration tool. URL: `http://swardtoolbox.github.io/`.