Put the cover here

APPROVAL SHEET

Title of Project: \_\_\_\_

Project Team: Mohamed

Aya

Esraa

Amira

Nada

Maha

Nora

Fatma

Computer and Systems Engineering, Class of 2018

Project Approved: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dr. Ahmed Amer Shahin

Assistant Professor

Computer and Systems Engineering

Date Approved: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ABSTRACT

|  |  |
| --- | --- |
| Title ofProject: | \_\_\_\_ |
|  |  |
|  | Ray7Gay Team, Class of 2018 |
|  |  |
| Directed By: | Ahmed Amer Shahin, Assistant Professor  Department of Computer and Systems Engineering |

Write the abstract here. Try to make it in one page or one and a half page.

|  |
| --- |
| PROJECT NAME |

By

|  |
| --- |
| Mohamed  Aya  Esraa  Amira  Nada  Maha  Nora  Fatma  \_ |

Project report submitted to the Computer and Systems Engineering

Department, Zagazig University, in partial fulfillment

of the requirements for the degree of

|  |
| --- |
| Bachelor of Computer and Systems Engineering |
| 2018 |

|  |
| --- |
| © Copyright by |
| Ray7Gay Team |
| 2018 |

# **Dedication**

We dedicate this to.

# **Acknowledgements**

First, all thanks are due purely to Allah, for providing us the blessings and the strength to complete this project. Second, we would like to express our deepest appreciation to \_\_\_\_.

# **Table of Contents**

[**Dedication** ii](#_Toc11688741)

[**Acknowledgements** iii](#_Toc11688742)

[**Table of Contents** iv](#_Toc11688743)

[**List of Tables** viii](#_Toc11688744)

[**List of Figures** ix](#_Toc11688745)

[Chapter 1: Preliminaries and Project Problem 1](#_Toc11688746)

[1.1 Introduction 1](#_Toc11688747)

[1.2 IoT Technology 1](#_Toc11688748)

[1.3 Smart Transportation 1](#_Toc11688749)

[1.4 Project Goals 2](#_Toc11688750)

[1.5 Project Contribution 2](#_Toc11688751)

[1.6 Organization 2](#_Toc11688752)

[Chapter 2: Overview 3](#_Toc11688753)

[2.1 System Architecture 3](#_Toc11688754)

[2.1.1 Customer 3](#_Toc11688755)

[2.1.2 Driver 3](#_Toc11688756)

[2.1.3 User 3](#_Toc11688757)

[2.2 The Reservation System 3](#_Toc11688758)

[2.2.1 Single Ride Reservation 3](#_Toc11688759)

[2.2.2 Registered Customers 3](#_Toc11688760)

[2.2.3 Payment Options 4](#_Toc11688761)

[2.3 Conclusions 4](#_Toc11688762)

[Chapter 3: Theories and Hardware 5](#_Toc11688763)

[3.1 Introduction 5](#_Toc11688764)

[3.2 Motion Control 7](#_Toc11688765)

[3.2.1 Kinematics 7](#_Toc11688766)

[3.2.2 Control Model 23](#_Toc11688767)

[3.3 Perception 34](#_Toc11688768)

[3.3.1 Sensors 34](#_Toc11688769)

[3.3.2 Sensors uncertainty: 35](#_Toc11688770)

[3.4 The Database 38](#_Toc11688771)

[3.5 Anything else \_\_\_\_\_\_ 38](#_Toc11688772)

[3.6 Conclusions 38](#_Toc11688773)

[Chapter 4: **theories** 39](#_Toc11688774)

[4.1 Bus Entry Gate 39](#_Toc11688775)

[4.2 Bus Exit Gate 39](#_Toc11688776)

[4.3 Cash Machine 39](#_Toc11688777)

[4.4 Conclusions 39](#_Toc11688778)

[Chapter 5: System Evaluation 40](#_Toc11688779)

[5.1 Evaluation Approach Overview 40](#_Toc11688780)

[5.2 Test Cases 40](#_Toc11688781)

[5.2.1 Test Case 1 40](#_Toc11688782)

[5.2.2 Test Case 2 40](#_Toc11688783)

[5.2.3 Test Case 3 40](#_Toc11688784)

[5.3 Conclusion 40](#_Toc11688785)

[Chapter 6: Conclusions and Future Work 41](#_Toc11688786)

[6.1 Summary of Contribution 41](#_Toc11688787)

[6.2 Future Work 41](#_Toc11688788)

[**References** 42](#_Toc11688789)

# **List of Tables**

**No table of figures entries found.**

# **List of Figures**

[Figure 1the see-think-act cycle 6](#_Toc11688790)

[Figure 2 The global reference frame and the robot local reference frame 10](#_Toc11688791)

[Figure 3Wheel configuration for a robot with differential-drive 15](#_Toc11688792)

[Figure 4 A detailed model of the differential-drive system 16](#_Toc11688793)

[Figure 5 Rotating the robot ωδt degrees around ICC 18](#_Toc11688794)

# Preliminaries and Project Problem

This chapter provides the motivation and necessary preliminary discussion about smart transportation systems and their use in making the life of people better. The chapter also highlights the main problem which is addressed in the project and summarizes our contribution.

## Introduction

Recent advances in IoT technology have been leveraged in many unconventional applications that facilitate and improve the safety and quality of people’s life in modern societies. An example of these applications is \_\_\_\_\_\_. \_\_\_\_\_\_.

However, most existing solutions for \_\_\_\_ either require \_\_\_, which may not be available, may incur extra charges, or suffer \_\_\_\_. In addition, there is no existing solution that allows \_\_\_\_. These shortcomings render existing solutions unsuitable for crowded locations like most of the cities in Egypt.

## IoT Technology

\_\_\_\_

## Smart Transportation

\_\_\_\_

## Project Goals

Write the goals that you want to achieve here.

## Project Contribution

The main contribution of this project is \_\_\_\_. The following highlights some of the features of the proposed solution:

* A
* B
* C

## Organization

This chapter has presented the motivation and necessary preliminaries for \_\_\_\_\_\_\_\_. Chapter 2 discusses \_\_\_\_\_. Chapter 3 covers \_\_\_\_. Chapter 4 presents \_\_\_\_. Chapter 5 explains \_\_\_\_. Chapter 6 concludes this project report with a summary of the contribution and outlines future research topics.

# Overview

In this chapter, we discuss our solution to the problem of \_\_\_\_\_. The system architecture and the interaction between different modules in the system is explained.

## System Architecture

The solution we proposed is composed of \_\_\_\_\_\_\_\_\_

### Customer

\_\_\_

### Driver

\_\_\_\_\_\_

### User

\_\_\_\_\_

## The Reservation System

Write some sentences \_\_\_

### Single Ride Reservation

\_\_\_\_

### Registered Customers

\_\_\_\_

### Payment Options

\_\_\_\_\_

## Conclusions

In this chapter, we have \_\_\_\_\_

# Theories and Hardware

## Introduction

An autonomous mobile robot is a robot that can do a variety of tasks with a high degree of autonomy (the ability to interact with the surrounding and take uninformed decisions to deal with different situations).

In autonomous mobile robots we need to answer three key questions:

1. Where am I ?
2. Where am I going ?
3. How to get there ?

And To answer these questions the robot has to have :

1. have a model of the environment (given or autonomously built -- map)
2. perceive and analyze the environment (perception).
3. find its position/situation within the environment (localization)
4. plan and execute the movement (planning and motion control).

So, autonomous mobile robots are mainly consist of four basic building blocks, each one of them have its own sensors, software and function.

These four blocks are called the see-think-act cycle and they are described as show below in the figure :

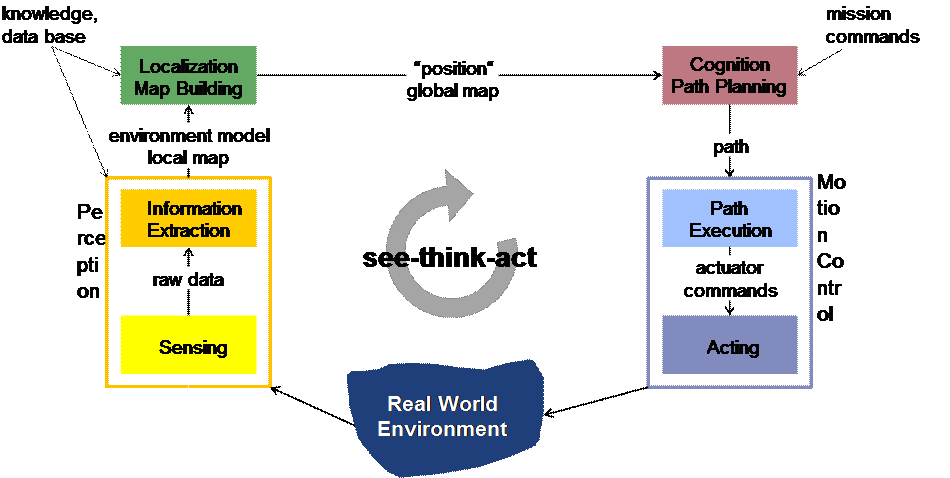


Figure 1the see-think-act cycle

In the coming sections we will describe each step of these four blocks in a great detail.

## Motion Control

Motion control is the first block of the see-think-act cycle and the remaining three blocks are built on above of it.

Motion control is divided to two main tasks:

1. Given a command to the robot to move to some place
2. Get continuous feedback from motion sensors to correct the robot path and to check is the robot has executed the commands correctly.

So, to implement these two tasks we need:

1. A motion model for the robot which is a kinematic model,
2. To build a control model using the kinematic model to control the robot motion.

### Kinematics

Kinematics is the most basic study of how mechanical systems behave. In mobile robotics, we need to understand the mechanical behavior of the robot both to design appropriate mobile robots for tasks and to understand how to create control software for an instance of mobile robot hardware.

In mobile robots we face the concept of the robot’s workspace which is the range of possible poses that the mobile robot can achieve in its environment. In mobile robots there is no direct way to measure a mobile robot’s position instantaneously.

Instead, one must integrate the motion of the robot over time. Add to this the inaccuracies of motion estimation due to slippage and it is clear that measuring a mobile robot’s position precisely is an extremely challenging task.

#### Kinematics Models and Constraints

Deriving a model for the whole robot’s motion is a bottom-up process. Each individual wheel contributes to the robot’s motion and, at the same time, imposes constraints on robot motion.

Wheels are tied together based on robot chassis geometry, and therefore their constraints combine to form constraints on the overall motion of the robot chassis. But the forces and constraints of each wheel must be expressed with respect to a clear and consistent reference frame. This is particularly important in mobile robotics because of its self-contained and mobile nature; so a clear mapping between global and local frames of reference is required.

#### Representing the position of the robot

As we are working in a horizontal 2D-Plane, the total dimensionality of this robot chassis on the plane is three, two for position in the plane and one for orientation along the vertical axis, which is orthogonal to the plane.

Of course, there are additional degrees of freedom and flexibility due to the wheel axles, wheel steering joints, and wheel castor joints. However, by robot chassis we refer only to the rigid body of the robot, ignoring the joints and degrees of freedom internal to the robot and its wheels.

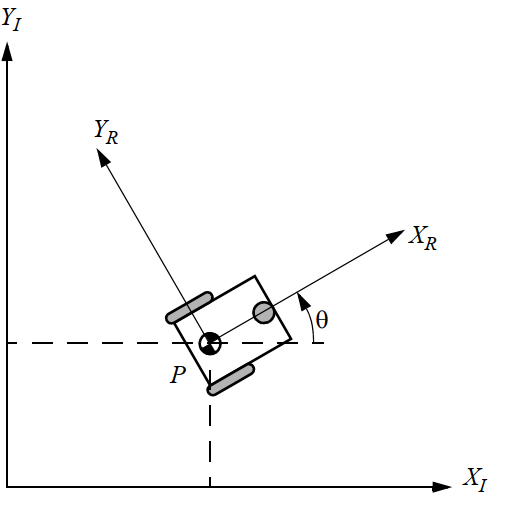
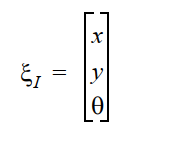


Figure 2 The global reference frame and the robot local reference frame

So in order to specify the position of the robot on the plane, we establish a relationship between the global reference frame of the plane and the local reference frame of the robot, as in figure 2.

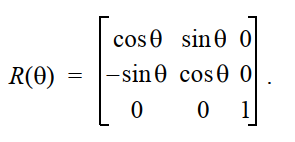
The axes and define an arbitrary inertial basis on the plane as the global reference frame from some origin O: {XI, YI }.

To specify the position of the robot, choose a point P on the robot chassis as its position reference point. The basis {XR, YR} defines two axes relative to P on the robot chassis and is thus the robot’s local reference frame. The position of P in the global reference frame is specified by coordinates x and y, and the angular difference between the global and local reference frames is given by 0 (an angle read as Theta ) . We can describe the pose of the robot as a vector with these three elements.



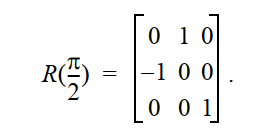
Note the subscript I is meaning that we refer to the global reference frame.

We can map between different reference frame using a matrix called orthogonal rotation matrix. This matrix is function of the current angle between the two frame (theta) and this matrix is defined as follow:



For example this matrix can be used to map motion in the global reference frame to motion in terms of the local reference frame using the equation below:

let theta = pi /2



So, given (x, y, theta) in the global frame we can map them to the robot local frame using the rotational matrix.

#### Forward Kinematics model:

In forward kinematics, given a set of actuators values e.g encoder ticks we plug these values in kinematic model equation to get the pose of the robot (x, y, theta).

In inverse kinematics, given the desired pose we want to go to, calculate the required actuators speeds and joints values to reach this pose if possible!!

We will use the differential drive model to reduce the complexity, cost, and size of the robot. A differential-drive robot consists of two main wheels mounted on a common axis controlled by separate motors. A differential drive system/steering system is a nonholonomic system, which means it has constraints on the pose change. A car is an example of a nonholonomic system, as it cannot change its position without changing its pose. Let's look at how our robot works and how we model the robot in terms of its mathematics.

A mobile robot or vehicle has six degrees of freedom (DOF) expressed by the pose (x, y, z, roll, pitch, and yaw). It consists of position (x, y, z) and attitude (roll, pitch, and yaw). Roll refers to sidewise rotation, pitch refers to forward and backward rotation, and yaw (called the heading or orientation) refers to the direction in which the robot moves in the x-y plane. The differential-drive robot moves from x-y in the plane, so the 2D pose consists mainly of x, y, and θ, where θ is the head of the robot

that points in the forward direction of the robot.

The forward kinematics equations for a robot with a differential-drive system are used to solve the following problem:

If robot is standing in a position (x, y, θ) at time t, determine the pose (x', y', θ') at t + δt given the control parameters V-left and V-right.

Where V-left (is the velocity of the robot left wheel), V-right (is the velocity of the robot right wheel). We will see later how to compute them.

This technique can be used in the robot to follow a particular trajectory.

#### Deriving forward kinematics equations:

Before deriving our model we will assume some assumptions:

1. Movement on a horizontal plane
2. Point contact of the wheels
3. Wheels not deformable
4. Pure rolling, no slipping, skidding, or sliding
5. No friction for rotation around contact point
6. Wheels connected to a rigid frame (chassis)

When the robot is about to perform a rolling motion, the robot must rotate around a point that lies along its common left and right wheel axes. The point that the robot rotates around is known as **ICC- Instantaneous**

**Center of Curvature.**

The following diagram shows the wheel configuration of differential-drive with ICC:

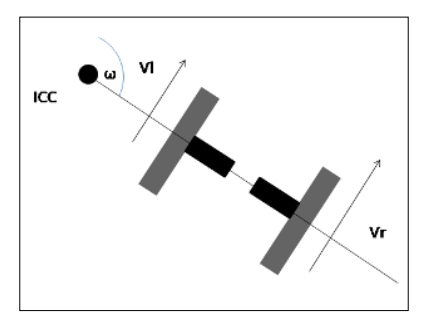


Figure 3Wheel configuration for a robot with differential-drive

The speed of the wheel is v = 2 πr / T, where T is the time taken to complete one full turn around ICC. The ω angular velocity is defined as 2 π / T and typically has the unit radians (or degrees) per second. Combining the equations for v and w yields:

(1)

A detailed model of the differential-drive system is shown in the following figure:

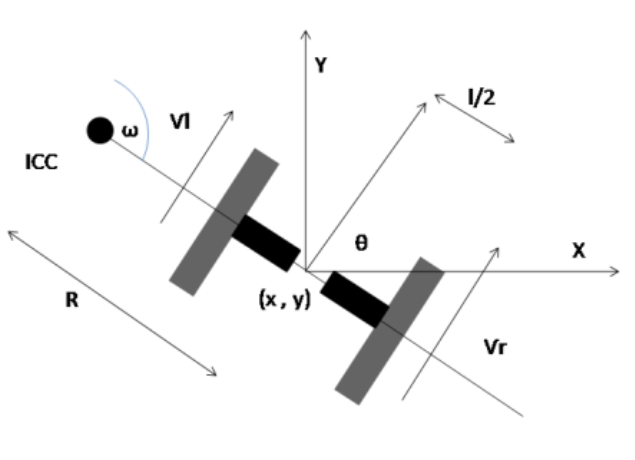


Figure 4 A detailed model of the differential-drive system

Where:

R: distance from ICC to the mid-point of the robot (center of the robot).

I: Width of the robot (distance between left and right wheels).

Vl, Vr: velocities of both left and right wheels.

W: angular velocity of the robot

Applying eq(1) on both wheels, the result will be :

(3)

After solving for we get the following results:

R = l/2 \* ( (Vl + Vr) / (Vr – Vl)) (4)

= (Vr – Vl)/l (5)

The previous two equations are useful for solving the forward kinematics problem. Suppose the robot moves with an angular velocity of *ω* for *δt* seconds, it can change the robot's orientation or where it is heading to:

θ′ = θ + \* δt (6)

and ICC will be given as, using simple trigonometry

ICC = [ ICCx , ICCy ] = [ x - Rsinθ, y + Rcos θ ] (7)

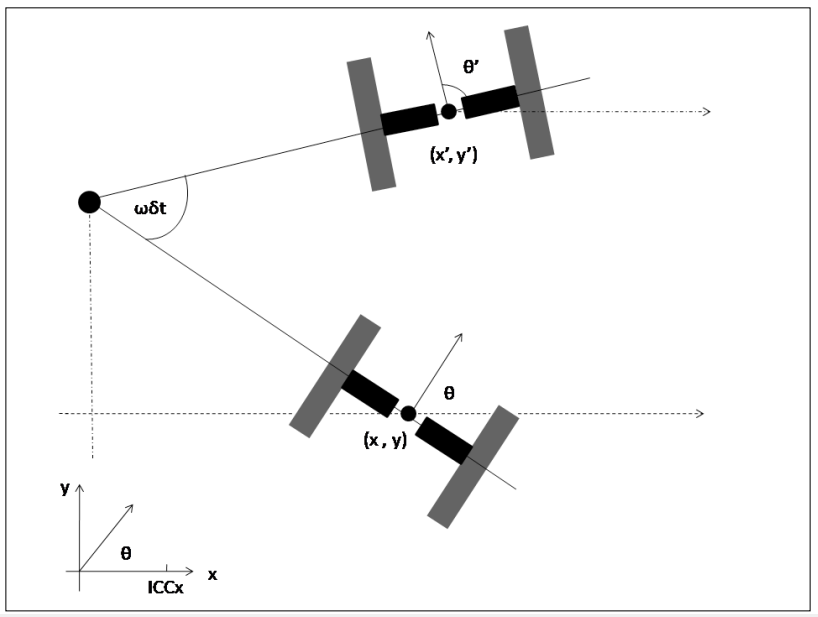


Figure 5 Rotating the robot ωδt degrees around ICC

**So, the overall conclusion** is that given a starting position (x, y), the new position (x',y') can be computed using the 2D rotation matrix (explained in the previous pages ). The rotation around ICC with angular velocity ω for δt seconds yields the following position at t + δt time:

#### 

The new pose (x’, y’, *θ'*) can be computed from eq 6 and 8, given *ω*, *δt*, and *R*.

ω can be computed from equation (5)**; Vr and Vl are often difficult to measure accurately**. Instead of measuring the velocity, the rotation of each wheel can be measured using a sensor called wheel encoders. The data from the wheel encoders is the robot's odometry values. These sensors are mounted on the wheel axes and deliver binary signals for each step the wheel rotates (each step may be in the order of 0.1 mm). These signals are fed to a counter such that v \* δt is the distance travelled from time t to t + δt. We can write:

n \* step = v \* δt

From this, we can compute *v*:

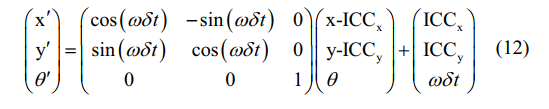
V = n \* step / δt (9)

If we insert equation (9) in equations (3) and (4), we get the following result:

R = ( l/2 \*(Vr + Vl) ) / (Vr – Vl) (10)

δt = (Vr – Vl)δt / l = (nr – nl ) \* step / l (11)

Here, nl and nr are the encoder counts of the left and right wheels. Vl and Vr are the speed of the left and right wheels respectively. Thus, the robot stands in pose (x, y, θ) and moves nl and nr counts during a time step δt; the new pose (x', y', θ') is given by:



Where R and δt are computed from eq 10,11 respectively and ICC is computed from eq 7

IMPORTANT NOTE:

The derived kinematic equation depends mainly on the design and geometry of the robot. Different designs can lead to different equations.

#### Inverse Kinematics

The forward kinematics equation provides an updated pose at a given wheel speed. We can now think about the inverse problem.

Stand in pose (x, y, θ) at time t and determine the V-left and V-right control parameters such that the pose at time t + δt is (x', y', θ').

In differential-drive, this problem may not have a solution because this kind of robot can't be moved to any pose by simply setting the wheel velocity. This is due to nonholonomic constraints.

In nonholonomic robots, there are some ways to increase the constrained mobility if we allow a different sequence (V-left, V-right). If we insert values from equations 7,10,11,12 , we can identify some special cases of control:

1. If V-right = V-left => nr = nl => R = ∞, ωδT = 0 =>: This means the robot moves in a straight line and θ remains the same
2. If V-right = -V-left => nr = -nl => R=0, ωδt = 2nl \* step / l and  
   ICC = [ ICCx , ICCy ] = [x, y ] => x' = x, y' = y, θ' = θ + ωδt =>: This means the robot rotates in the position around ICC, that is, any θ is reachable, while (x, y) remains unchanged

**The Result of the above two conclusions:**

Combining these operations, the following algorithm can be used to reach any target pose from the starting pose:

1- Rotate until the robot's orientation coincides with the line from the starting position to the target position, V-right = -V-left = V-rot.

2- Drive straight until the robot's position coincides with the target position, V-right = V-left = V-ahead.

3- Rotate until the robot's orientation coincides with the target orientation, V-right = -V-left = V-rot.

where, V-rot and V-ahead can be chosen arbitrarily

For more details on differential robot kinematics, refer to:

1. [Control of mobile robots, coursera](https://www.coursera.org/learn/mobile-robot)
2. Introduction to autonomous mobile robots

[Book](https://mitpress.mit.edu/books/introduction-autonomous-mobile-robots-second-edition) chapter 3, [course](https://www.edx.org/course/autonomous-mobile-robots-2) week 1 and week 2

1. [Slam lectures](https://www.youtube.com/playlist?list=PLpUPoM7Rgzi_7YWn14Va2FODh7LzADBSm) unit A

### Control Model

Why we need control?!

1. The objective of a kinematic controller is to follow a trajectory described by its position and/or velocity profiles as function of time.
2. Motion control is not straight forward because mobile robots are typically nonholonomic and MIMO systems.
3. Most controllers (including the one presented here) are not considering the dynamics of the system

#### PID Controller

PID Controller is basically a method used in programming and if tuned properly, can be incredibly effective and accurate. PID stands for Proportional Integral Derivative, 3 separate parts joined together, though sometimes you don't need all three. For example, you could instead have just P control, PI control or PD control.

**A bit of history...**

PID controllers date to 1890s governor design. PID controllers were subsequently developed in automatic ship steering. One of the earliest examples of a PID-type controller was developed by Elmer Sperry in 1911, while the first published theoretical analysis of a PID controller was by Russian American engineer Nicolas Minorsky. Minorsky was designing automatic steering systems for the US Navy, and based his analysis on observations of a helmsman, observing that the helmsman controlled the ship not only based on the current error (distance/value remaining), but

also on past error and current rate of change; this was then made mathematical by Minorsky. His goal was stability, not general control, which significantly simplified the problem. While proportional control provides stability against small disturbances, it was insufficient for dealing with a steady disturbance, notably a stiff gale (due to droop), which required adding the integral term. Finally, the derivative term was added to improve control.

– Adapted from [Wikipedia.](https://en.wikipedia.org/wiki/PID_controller#History_and_applications)

PID controllers are used widely in industry. In fact, 95% of all closed loop systems in industrial processes are run off PID control.

So, now we know a brief background, let's get on with explaining it all.

**P – Proportional**

Imagine a robot that travels at full speed, for let's say, a value reading 1000 on the sensor used. Now, because of it's speed and inertia, it will probably overshoot a little and travel further than 1000 on the sensor. This can be a real nuisance when writing a program, as you want as much accuracy as possible. In a perfect world, you would be able to tell the robot to stop and it would stop exactly where it was...

However, we aren't in a perfect world, and we have overshooting problems if we tell it to stop suddenly. By telling it to stop suddenly,it will overshoot

**Now this overshoot wouldn't be a problem if the distance it overshot was always the same.**

However, there are lot's of variables that can change the distance it overshoots. Here are a few of the variables:

• Battery voltage. If the battery is low, the motors won't run as fast and so the robot will have less inertia. In this case, the robot will overshoot less.

• If the robot hits something, the overshoot will become less.

• If something pushes the robot in the direction it wants to travel, the overshoot will become greater.

So as you can see, overshoot is not good. So the P controller controls the speed smoothly, allowing it to slow down as it approaches it's target, to shrink the overshoot. **That's why it is called a proportional controller – the output speed is proportional to the value remaining to be changed, which we call an error, where error = (target value – current value).**

However, P-controller alone is not always enough to do the task as it has some problems:

1. It cannot get rid of steady state error , you will always have some error so the controller can work.
2. The controller is proportional to error value, so if error is high => there will be a bigger overshoot and vice versa

As a result in most applications we need some extra control, this can be achieved using I and D controllers

**I – Integral**

So the proportional part of the code has got it so that the error remaining is pretty small. Too small for the proportional section to make much of a difference. This is where the integral comes in. The integral is the running sum of previous errors. So when your error is very small, the integral comes into action, but how does it actually work?

The integral wants to get it so that it travels fast enough to shrink the error, but not too fast, because then it might run the risk of overshooting. The way it decides how fast to go is that it will gently accelerate. The integral can be calculated like this:

**integral = integral + error\*dT;**

dt here the time between two continuous sample i.e the sampling rate .

**problems with the integral**

**problem 1**

When your error finally reaches 0, your integral will probably still be at a value which keeps the speed high enough to keep the error changing. The equation will only reach 0 by itself if it passes past an error of 0, so the negative error can subtract into the existing integral. So, if the speed is still high enough to keep the error changing, we have a problem, right? There is a very simple solution to this problem, and that is to reset the integral if the error reaches 0. This can be done as follows. The fix in the code is in bold text

**if (error is 0)**

**{**

**integral = 0;**

**}**

**Problem 2:**

It is known as integral wind-up. It can start by if you have a large error to travel, the integral will start to build up once the loop starts to run. So, by the time the integral needs to be used, it is already at a value far higher than is usable. There are simple solutions to this issue to, **2 of which I will address**

1. **Limit the value of the integral**

Limit the value that the integral can reach. If it is reaching too high, why not just put a limit on it? A limit could be written as follows:

if (integral is greater than or equal to the maximum value)

{

integral = maximum value;

}

But, if the integral is too big but in it's negative form (I. E. making the speed reverse too fast), you would need to rewrite the same as above but for the negative version of the integral.

1. **Remove pid windup and make controller output maximum**

/// integral wind up removal

if(control\_signal > max\_val){

control\_signal = max\_val;

integral = integral - (error \* ki);

}

if(control\_signal < min\_val){

control\_signal = min\_val ;

integral = integral - (error \* ki);

}

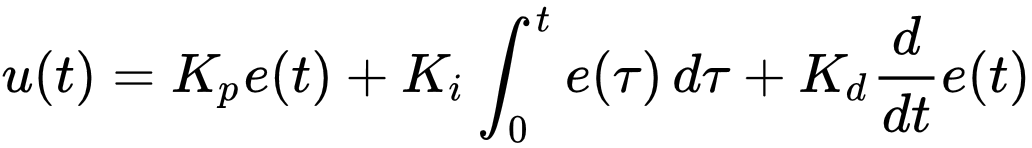
**D – Derivative**

The final bit of the PID code – the derivative! The job of the derivative is to predict the future value for the error, and then make the speed act accordingly. For example, if it thinks it will overshoot, it will slow it down

derivative = d error / dT

= ( **(current error) – (previous error)** ) / dT

So the overall control signal is calculated using the following eq:



For full PID implementation code see our code in this file:

GPEditor-master\src\baymaxrobot\arduino\firmware\lib\pid

**Tuning the constant terms**

This is the most time consuming and labor intensive. There are many different methods for tuning the Kp, Ki and Kd, I will try to explain the one we have used in our project as best as I can. Ways to tune the PID constants can be done by a computer program, by math calculations or by

manual tuning. I highly recommend at all times to view all the values for the error, speed etc. so you can see how close it is getting to the target point/how much is remaining to be changed. Use a debugger or a similar monitoring tool to check the results.

**Some concepts you need to understand them very well for better tuning:**

**Rise time** – the time it takes to get from the beginning point to the target point for the first time.

**Overshoot** – the amount that is changed too much; the value further than the error.

**Settling time** – the time it takes to settle back down when encountering a change i.e the time the output takes to be in the range of 95 – 98 of the final value.

**Steady-state error** – the error at the equilibrium

**Stability** – the “smoothness” of the speed

**What happens when each of the constants is increased:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Constant:** | **Rise time:** | **Overshoot:** | **Settling time:** | **Steady-state error:** | **Stability:** |
| **Kp** | decrease | increase | Small change | decrease | degrade |
| **Ki** | decrease | increase | increase | decrease | degrade |
| **Kd** | minor change | decrease | decrease | No effect | Improve (if small enough) |

**PID Tuning (manual tuning):**

The way I tune my constants is as follows:

1. Set Kp, Ki, and Kd to 0. This will disable them for now.

2. Increase Kp until the error is fairly small, but it still gets from the beginning to nearly the end quickly enough.

3. Increase Kd until any overshoot you may have is fairly minimal. But be careful with Kd – too much will make it overshoot.

4. Increase Ki until any error that is still existing is eliminated. Start with a really small number for Ki, don't be surprised if it is as small as 0.0001 or even smaller.

5. Using the rules of tuning the constants (in the table on the previous page), you can change around the constants a little bit to get it working to the best performance.

Knowing how big/small each constant term should be gets easier with practice, you will get a rough idea for it after practicing a few times.

There is a **second** and good method for tuning which is **Ziegler-Nichols method** we do not use it in our project but I will explain it if you want to use it in any project.

In this method, the values for the constant terms are mainly found out by math, **but you will almost definitely need to tweak them a bit afterwards, to get it all working perfectly. The first bit of this method is done manually, but after you have done a few tests it is on with the math.**

1. **Set Kp, Ki and Kd to 0**. This will disable them for now.

2. Start **increasing Kp until there is some oscillation in the error**. Not too much oscillation, just a noticeable amount. Save this value for Kp as **“Ku” - the value for the ultimate or critical gain**.

3. **You now need to measure the period of the oscillation**. Save the value for the period as “Pu” - the period at the ultimate or critical gain.

4. Now we have done the most of the manual part of this method, so we will use the math of finding Kp, Ki and Kd!

There are 3 “constroller types”. P is a simple proportional controller, PI is proportional with integral, PID is the full proportional integral derivative code. There's no PD (proportional with derivative) in the table, but that doesn't say it doesn't exist. Here is the table for finding Kp, Ki and Kd:

**Ziegler-Nichols Method for finding Kp, Ki and Kd:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Controller type:** | **Kp:** | **Ki:** | **Kd:** |
| **P** | 0.50Ku | N/A | N/A |
| **PI** | 0.45Ku | 1.2Kp/Pu | N/A |
| **PID** | 0.60Ku | 2Kp/Pu | KpPu/8 |

As I mentioned before, you will probably want to adjust and tweak the values for Kp, Ki and Kd after the calculations, as they might not be completely perfect for you scenario. Use the rules for finding the constant terms to aid you.

For more details refer to:

1. [WikiPedia](https://en.wikipedia.org/wiki/PID_controller)
2. [Control of mobile robots, coursera](https://www.coursera.org/learn/mobile-robot)

## Perception

Perception is the second block is the see-think-act cycle and its mainly concerned with perceiving the surroundings using various sensors and filter the perceived data then analysis the filtered data to extract meaningful information.

### Sensors

A wide variety of sensors is used in mobile robots. Some sensors are used to measure simple values such as the internal temperature of a robot’s electronics or the rotational speed of the motors. Other more sophisticated sensors can be used to acquire information about the robot’s environment or even to measure directly a robot’s global position.

Sensors are classified to two main types:

proprioceptive/exteroceptive and passive/active

**Proprioceptive sensors:**

Sensors that measure values internally to the system (robot),

e.g. motor speed, wheel load, heading of the robot, battery status

**Exteroceptive sensors:**

Sensors that extract information from the robot’s environment

e.g. distance to object, intensity of the ambient light, unique features.

### 3.3.2 Sensors uncertainty:

Sensors are imperfect devices with errors of both systematic and random nature. Random errors, in particular, cannot be corrected, and so they represent atomic levels of sensor uncertainty.

But when we build a mobile robot, we combine information from many sensors, even using the same sensors repeatedly, over time, to build, possibly, a model of the environment. How can we scale up, from characterizing the uncertainty of a single sensor to the uncertainty of the resulting robot system?

In our project we have used the most common uncertainty representation model which is **the standard Gaussian uncertainty model to model our sensors uncertainty, we built a model for each sensor alone.**

Then, after that we use what is known Kalman filter to merge these model and get the best approximation for our robot position.

Perception is mainly concerned with computer vision and image processing and our project has a portion concerned with computer vision so I will let this part to my colleague.

**But for more details on this part , I highly recommend reading chapter 4 of this** [Book](https://mitpress.mit.edu/books/introduction-autonomous-mobile-robots-second-edition)

## SLAM

\_\_\_\_\_

## Anything else \_\_\_\_\_\_

\_\_\_\_\_\_

## Conclusions

In this chapter, we have presented \_\_\_\_

# **theories**

In this chapter, we discuss the various components we used to build the bus entry and exit gates, the cash machine, …...

## Bus Entry Gate

\_\_\_\_\_\_\_

## Bus Exit Gate

\_\_\_\_\_

## Cash Machine

\_\_\_\_\_\_\_

\_\_\_\_\_\_

## Conclusions

In this chapter, we have presented\_\_\_\_\_\_

# System Evaluation

In this chapter, we discuss how we have evaluated our proposed system. In addition, we present the results of real test scenarios that we have carried out.

## Evaluation Approach Overview

\_\_\_\_\_\_

## Test Cases

We have carries out several test scenarios such as\_\_\_

### Test Case 1

\_\_\_\_\_\_

### Test Case 2

\_\_\_\_\_\_

### Test Case 3

\_\_\_\_\_

## Conclusion

In this chapter, we have presented \_\_\_\_

# 

# Conclusions and Future Work

Advances in IoT technology have made \_\_\_\_. This chapter summarizes the contribution of the project and outlines the planned future work.

## Summary of Contribution

In this project, we \_\_. The following is a summary of the specific contributions:

1. *\_\_\_*
2. *\_\_\_*
3. *\_\_\_\_*

The effectiveness of our solution is validated through \_\_\_\_. The results have confirmed the advantages of our system in terms of \_\_\_.

## Future Work

As we pointed out, the aim of this project is to \_\_\_\_. In the future, we plan to further extend our work by \_\_\_. We plan also to \_\_. Finally, expanding our work to other platforms, like \_\_\_.

# **References**

1. Betts, B.; et al., “Improving situational awareness for first responders via mobile computing. NASA Ames Research Center,” *Smart Systems Research Laboratory*, 2006.
2. Boddhu, S. ; et al.,"Increasing situational awareness using smartphones", *Proc. SPIE 8389, Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR III*, 83891J (May 1, 2012)