Brock University
Faculty of Mathematics & Science
Department of Computer Science

COSC 4P78: Robotics

4P78 Project Documentation

Prepared By:

Parker TenBroeck 7376726
pt21zs@brocku.ca

Brett
brett@brocku.ca

Instructor:
Earl Foxwell

April 26, 2025

Contents

1 Introduction **2**

2 Instructions **2**

3 Problem Set **2**

3.1 Balancing 2

3.2 Odometry 2

3.3 Target Positions 3

3.4 Communication 3

4 Approaches **3**

4.1 Balancing 3

4.2 Odometry 3

4.3 Target Positions 3

4.4 Communication 3

5 Challanages **3**

5.1 Balancing 3

5.2 Odometry 3

5.3 Target Positions 4

5.4 Communication 4

6 Resources Used **4**

1 Introduction

What if you wanted to make a robot to map out a room but you only had two wheels and two motors at your disposal? Well we have the perfect solution for you! Introducing the Bobot, a two wheeled robot which is more of a circus act than useful.
With only two wheels its purely dynamically stable, will fall over due to its own stupidity but its very cute while it does so! This little guy can wiz around at the speed of a snail your house and (poorly) map out a room to your hearts desire

2 Instructions

- Hold robot in a vertical upright position clear of obsticals
- Place battery in top compartment
- Plug battery in and wait for robot to initialize
- Once robot has booted let go and step away
- Connect computer to network `MEOW`
- launch vidualization and control software

3 Problem Set

3.1 Balancing

We have a vertical two wheeled robot I hope its fairly obvious when I say that it is only dynamically stable. This presents a challance because we need some system which keeps the system upright in a variaty of situations and responds dynamically to changes it cannot predict.

3.2 Odometry

Knowing that our system is very dynamic and having the forsight to know that we will need to keep in constant motion to keep upright the odometry system will need to handle that.

3.3 Target Positions

The problem of target positions like odometry is made more challenging due to the fact we are almost constantly in motion. We simply cannot ”not move” when we are in the location we want to be as we'd fall over. This means the target position will need to be a ”best effort” battle to keep in the same place as best as we can.

3.4 Communication

The only source of computation used is a single `ESP8266` microcontroller, this in combination with the strict timing requirements on the duration of the control loop means the communication between the robot and the control/mapping software needs to be quick and efficient.

4 Approaches

4.1 Balancing

4.2 Odometry

4.3 Target Positions

The target position

4.4 Communication

5 Challanages

5.1 Balancing

Obviously a robot which stays upright and only has two wheels

5.2 Odometry

Keeping track of our position and angle was another challange that required careful consideration. Because the system is so dynamic and in constant motion we needed a system which could account for the constant movement to maintain a stable state.

5.3 Target Positions

Because the robot is in constant motion keeping itself balanced without a ”push” towards a single position it will drift around. To solve this we use the odometry system as a input to the movement system. By setting the desired heading of the robot to the vector from its position to the target position, and by biasing the direction the robot will travel to be the direction to the target position we get a crude way of staying in a single position.

5.4 Communication

Since we have limited processing power and time per loop iteration we need to be smart in how we receive and transmit data to our mapping software. For this reason we designed a stateless UDP based network protocol overtop the esp8266 Wifi & UDP libraries. [1]

6 Resources Used

- ESP8266 core libraries to note the Wifi and UDP libraries. [1]
- `PID_V1` The PID library of choice. [2]
- Adafruit VL53L0X was used the library used to interface with the time of flight sensor used for mapping. [3]
- The AS5600 library was used to interface with the AS5600 magnetic encoders on each wheel. [4]
- The MPU6050 library was used to interface and interpret the accelerometer and gyroscope data. [5]. It required modification to work with our hardware as it was a knockoff and the device ID was different than what it was expecting.
- A blog by the author of the PID library used was very helpful when tuning and configuring the PID controls in the robot. [6]
- A paper on odometry for robots with differential steering which we based our odometry system off of. [7]

References

[1] "Arduino core for ESP8266 WiFi," https://github.com/esp8266/Arduino.

[2] Brett Beauregard, "Arduino PID Library," https://github.com/br3ttb/Arduino-PID-Library/tree/master.

[3] "Adafruit VL53L0X Library," https://github.com/adafruit/Adafruit_VL53L0X.

[4] "AS5600 Library," https://github.com/RobTillaart/AS5600.

[5] Electronic Cats, "MPU6050 Library," https://github.com/ElectronicCats/mpu6050.

[6] Brett Beauregard, "Introducing Proportional On Measurement," http://brettbeauregard.com/blog/2017/06/introducing-proportional-on-measurement/.

[7] G.W. Lucas, "A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators," https://rossum.sourceforge.net/papers/DiffSteer/.