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

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
- Pan with WASD and zoom with scroll or +/-
- Control by clicking on screen to set target position
- use buttons/dropdowns to set/zero/get values from the robot

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

Because we intend to map an area with the robot we need to know where we are. This issue is made slightly more challanging due to the fact we are in nearly constant motion keeping ourself upright.

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

The balancing is done by

4.2 Odometry

4.3 Target Positions

The target position is acheived through the combination of two PID controllers. One is responsible for turn and simply adds motor speeds directly to the output of the balancing PID controller. One thing to note is the max turn speed is only 15% of the max motor speed and decreases linearly as the angle goes to $\pm 12^\circ$ to increase stability. The second PID controller is responsible for moving forward/backwards. It works by being the input to the balancing PID controller and setting the angle setpoint. it is bound to $\pm 2^\circ$ and will bias the

robot to move in one direction.
These work together by calculating the heading and displacement needed from the robots current position to the target position and setting those values as the input to the aforementioned PID controllers.

4.4 Communication

Communication is done through a stateless but sequenced and tagged UDP

5 Challanages

5.1 Balancing

Balancing took the longest out every task combined. It required research on not only PID control loops but also on different libraries and calibration requirements for our `MPU6050` gyroscope. These difficulties were compounded by the fact that we did not know which part/section would cause the robot to "randomly flail about and violently crash into the wall/floor".

5.2 Odometry

We origionally didn't have encoders to keep track of our position and instead attempted to use the accelerometer to calculate displacement/heading to save money on parts. This however, did not pan out. The first problem we ran into was our heading drifting. We knew it was going to be an issue after reading docs as the module does not have a magnetometer builtin. That aside we tried a good old fassion "how bad could it possibly be" and oh boy did it drift fast. The second issue was "double integration over a noisy inaccurate signal" lead to our calculated position quickly exiting the stratosphere. These issues very quickly lead us to just get encoders.

5.3 Target Positions

Using the bias towards a direction by setting the target angle was not the first approach we used. We originally directly added a movement speed to the motors like we do for turning. This however caused the robot to become unstable in certain situations. We eventually abandoned the approach in favor of the one used currently.

5.4 Communication

Since we have limited processing power/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]

We originally used a TCP+Webserver based library for this as it allowed us to use HTML+JS to display information but it continually had processing requirements the controller could not meet.

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/.