Driving in a Straight Line accurately

Böhler Fabian, Fellner Christoph, Griesmayer Verena, Groiß Simon, Kawicher Sebastian
Department for Computer Science
Secondary Technical College
Wiener Neustadt, Austria
Email: anubis.airlab@gmail.com

***Abstract*—Driving in a straight line with the KIPR Wallaby is a challenge. Therefore, we compared six different methods for accurate driving. This includes multiple programs using a Kalman filter and a PID controller to manipulate either the compass values or the motor ticks. We used a simple hardcoded method as a reference to evaluate the efficiency. The goal of this paper is to determine which of these methods is the most accurate by comparing them in an experiment.**

I. Introduction

While practicing for the Botball contest we noticed that for driving a certain distance, hardcoding is not always the best solution due to its inaccuracy. We thought about using a Proportional-integral-derivative (PID) controller to improve accuracy. We asked ourselves which implementation of various PID controllers is the most precise and if a Kalman filter can be used to maximize precision.

A. Hardware

Fig. 1. The controller includes 10 digital sensor ports, 6 analog sensor ports, 4 motor ports and 4 servo ports. There is also a built in compass.

The controller system used in the experiments in this paper is KIPR Wallaby. See figure[1].

II. Kalman filter

The Kalman filter [2] is a technique for estimating values by multidimensional normal distributions of erroneous observations. In this case the Kalman filter is used to reduce the errors of the measurements. After each new measurement, the Kalman filter improves the previous estimates and updates the associated error estimates.

The Kalman filter is used in different industries, such as the computer industry and spacecraft industry.

III. PID controller

The PID controller (Proportional-Integral-Differential controller)[3] is a transmission element of control engineering. It is composed of parts of a "P" component, an "I" component and a "D" component. Although PID controllers are hard to accurately calibrate, they do offer some advantages. The P element ensures a fast response, the I element can completely compensate for the control deviation, and the D element enables fast readjustment in the event of sudden disturbances. The formula of the PID controller can be seen in the equation 1.

$$u(t) = \overbrace{K_p e(t)}^{Proportional} + \overbrace{K_i \int_o^t e(\tau)d\tau}^{Integral} + \overbrace{K_d \frac{d}{d_t} e(t)}^{Derivative} \quad (1)$$

Fig. 2. PID equation

A. P component (Proportional)

The P component consists exclusively of a proportional part of the gain Kp. With its output signal u it is proportional to the input signal e. The P component will also trigger an immediate change of the manipulated variable, but its effect is initially not as dominant as that of the D element. On the other hand, the proportional effect remains constant in time.

B. I component (Integral)

An I component acts by time integration of the control deviation $e(t)$ on the manipulated variable by weighting the reset time. With increasing duration, the part of the I element comes into play which controls the system without permanent control deviation.

C. D component (Derivative)

The D component is a differentiator which is only used as a controller in connection with P and/or I controllers. It does not react to the amount of the control deviation e(t), but only to its rate of change. If there is an abrupt change in the control deviation, the control of the D element (which reacts to the change in the control deviation) comes to the fore and ensures a rapid reduction in the control deviation.

IV. Implementation

For measuring the deviation we used a software which detects the Aruco marker[4] on our bot, as shown in figure 4. The values of the different recordings have been recorded and visualised in two Graphes, one for visualising the deviation in Centimeters and one for visualising the deviation in degrees. For testing, we compared six different algorithms.[1]

A. Hardcoded

First we implemented the classic hardcoded variant. As the motors by themselfs are not very accurate, instructing the robot to drive a straight line will result in the robot diverging from its intendet path. As the hardcoded variant does not compensate for this error, the robot will sigificantly stray away from its path.

[1]The corresponding code is available at: [5]

B. PID on motor values

To improve accuracy we used the ticks of the motor in combination with a PID controller to estimate which one is turning faster and balance the driving.

C. PID on motor values + Kalman filter

To get even more precision we used a Kalman filter together with the PID. The code can be seen in 1

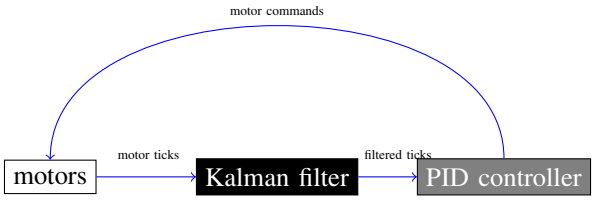

Fig. 3. Overview of communication between components

```
// P component
double P = 15.0;
// I component
double I = 1.0;
// D component
double D = 0.5;

// Initialisation
// of the PID controller
PID pid = PID(&error,
              &out,
              P,
              I,
              D);

// Processed motor ticks
// with a Kalman filter
double rd = filter(R_ticks());
double ld = filter(L_ticks());
error = ld - rd;

pid.Compute()

power(lMotor, speed + out);
power(rMotor, speed - out);
```

Listing 1: example code of implemented approach C using the motor ticks and a PID controller

D. compass with a PID controller

We also implemented a program that uses the orientation of the built in compass to adjust the rotation of the robot while driving. We used PID controller to get more accurate values. We expect this to result in even more precise driving than moving with motor ticks.

E. compass with a PID + Kalman filter

Furthermore we wanted to know if the accuracy will improve, if we process the compass values with a Kalman filter. We expect this to work better than without a Kalman filter.

F. PID + Kalman filter on motors and compass

At last we used both, the compass and the motor values as reference. We expect this to be the most accurate implementation because of the use of both the compass and the motor ticks.

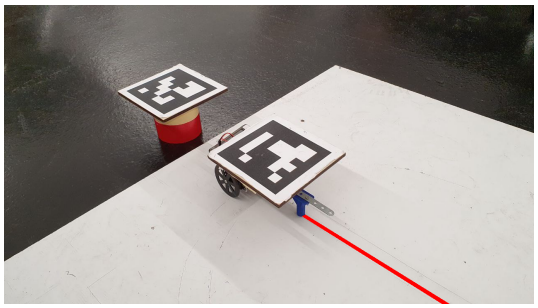

Fig. 4. Starting position of the robot, next to the stationary Arucotag, which enables measuring the deviation from the straight path(in red)

V. Experiment

For testing we used a robot with an Aruco marker on top and a second stationary one as seen in figure 4. We use these markers to track the movement of the robot. We then started the programs we wanted to test. During these experiments another program recorded the positions of the robot every 33 milliseconds using the video-stream of a camera. With these positions we evaluated the deviation in cm and degrees and transferred these values into the graphs in figure 5 and figure 6.

```
// get the reference angle
// for the compass
double start = get_compass();

// [...]

double speed = 70;
// P component
double P = 0.8;
// I component
double I = 1.0;
// D component
double D = 0.5;

// Initialisation
// of the PID controller
PID pid = PID(&error,
              &out,
              P,
              I,
              D);

// Threshold for the PID
// controller to kick in
double th = 5.0;
double angle = get_compass();

// Calculate the difference
// between the current and
// the starting angle
error = angle - start;

if (diff < -th ||diff > th) {
    pid.Compute()
    power(lMotor,speed + out);
    power(rMotor,speed - out);
} else {
    power(lMotor, speed);
    power(rMotor, speed);
}
```

Listing 2: example code of approach D using the compass with a PID controller

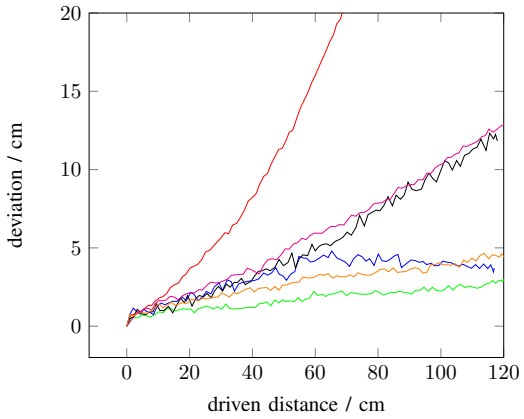

Fig. 5. Graph depicting the deviation in Centimeters from the straight path during the various experiments

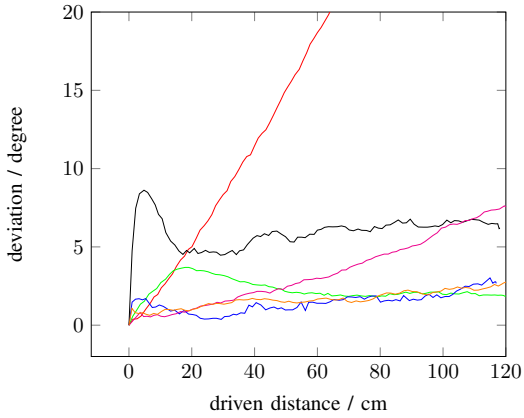

Fig. 6. Graph depicting the deviation in degrees from the straight path during the various experiments

Legend:

- hardcoded
- compass with a PID
- PID + Kalman filter on motors and compass
- compass with a PID + Kalman filter
- motors with a PID
- PID + Kalman filter on motors

VI. Conclusion

As seen in figure 5 and figure 6, the hardcoded way to drive straight performed the worst, due to its high inaccuracy as expected. Furthermore, driving only with a PID controller on the motor values is also not very accurate. The difference between using the compass with PID and using the compass with PID and Kalman filter was greater than expected. Using a PID controller, compass and Kalman filter surprisingly gave the third worst result, while using a compass with only a PID controller turned out to be the most accurate method. However, efficiency is not only about accuracy but also about the reliability of a code. Since any algorithm with the compass is inconsistent and laborious to calibrate compared to the other methods, the variant with a PID controller and the compass is the most accurate but not the most efficient. Due to our findings we conclude that, the most efficient method is using the PID controller in combination with a Kalman filter processing the motor ticks, because the difference in accuracy to a method with compass and PID is relatively small but much more reliable.

VII. Acknowledgment

The authors would like to thank Dr. Michael Stifter for the great support in realizing this project. We would also like to thank all the roboticists who helped us with this paper, special thanks to Joel Klimont who helped us implementing the software we used for the measurements in the experiment.

References

[1] KIPR, *Wallaby Controller*, https://www.kipr.org/kipr/hardware-software, store page, accessed March 18th 2022

[2] *Kalman filter*, https://perso.crans.org/club-krobot/doc/kalman.pdf, information page, accessed March 18th 2022

[3] *PID*, https://www.researchgate.net/profile/Robert-Paz-2/publication/237528809_The_Design_of_the_PID_Controller/links/004635360fa1ebdf63000000/The-Design-of-the-PID-Controller.pdf, information page, accessed Januar 14th 2022

[4] *Aruco marker*, https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html, information page, accessed March 18th 2022

[5] *GitHub Repository*, https://github.com/DuSack1220/anubis_paper_2022 source code, accessed March 18th 2022