Advanced Robot Positioning
Long Johannes Wang and Lisamarie Schuster
PRIA Allstars
johannes.wang@live.at

Advanced Robot Positioning

How to use the EMF feedback of Black Gear Motors to navigate on the KIPR Link

1. Introduction

Navigation in a scientific context is a field of study which covers the process of monitoring and controlling the movement of an object. [1] These movements are desired to be fast, accurate and repeatable. These requirements cause many problems.

As a matter of principle, the accuracy of a movement decreases with increasing speed if there is not a proper control loop to counter the rising error, thus making robots with high velocities hard to control. Therefore, precise motions need to be executed in a slow manner. Due to these facts it is impossible to be fast and accurate at the same time.

Even though it might be possible to have a robot drive a short defined track with a more or less high velocity, the number of consecutive motions increases the total error at the end of the whole motion. The repeatability depends on the accuracy of each motion since the error of a preceding motion is generally carried into the next movement because the end position of the previous action is the start position of the next one. Due to this cumulative error it is necessary to realign a robot in periodic intervals or whenever possible.

In Botball, navigation is about using actuators to move a robot from one spot to another, usually necessary to utilize some kind of effector at the destination to score points. Therefore it is crucial to have a high success rate of reaching the destination within the error margin.

2. The problem of common steering

The thing referred to as common steering in this paper is the method used by the average Botball team. Be it a newcomer or a veteran team with 6 years' experience, most participants use approximately synchronous, separate control algorithms to steer. These usually work by setting the motor powers or velocities to a desired value at the initiation of a motion and then cancelling the movement by stopping the motors when certain criteria are fulfilled. Some examples for these criteria are passing a certain amount of time, reaching a defined sensor value or exceeding a defined motor position. These methods have sufficient accuracy for most applications and by regularly realigning the robot it is possible to achieve high success rates. However, realignments need time and as mentioned above this kind of steering loses accuracy with increased execution speed. Therefore the team needs to find the golden line between speed and accuracy to obtain the highest possible execution speed while maintaining a sufficient success rate.

There are two major flaws that cause the inaccuracy of these methods of steering. First and foremost, since the separate movements of a program are not linked in any way and do not have any information about previous movements the robot is unable to determine the error that has occurred up to the point at which it is standing. The second issue is that the two motors are controlled individually. The typical way to minimize error while steering is to use two separate PID controllers which control the two motors. These PID controllers try to maintain a given speed which is set by the programmer in the program. The problem in doing so is that the PID controller of one motor does not know the velocity the other motor turns at. Due to that fact, neither of the motors can react to errors made by the other motor.

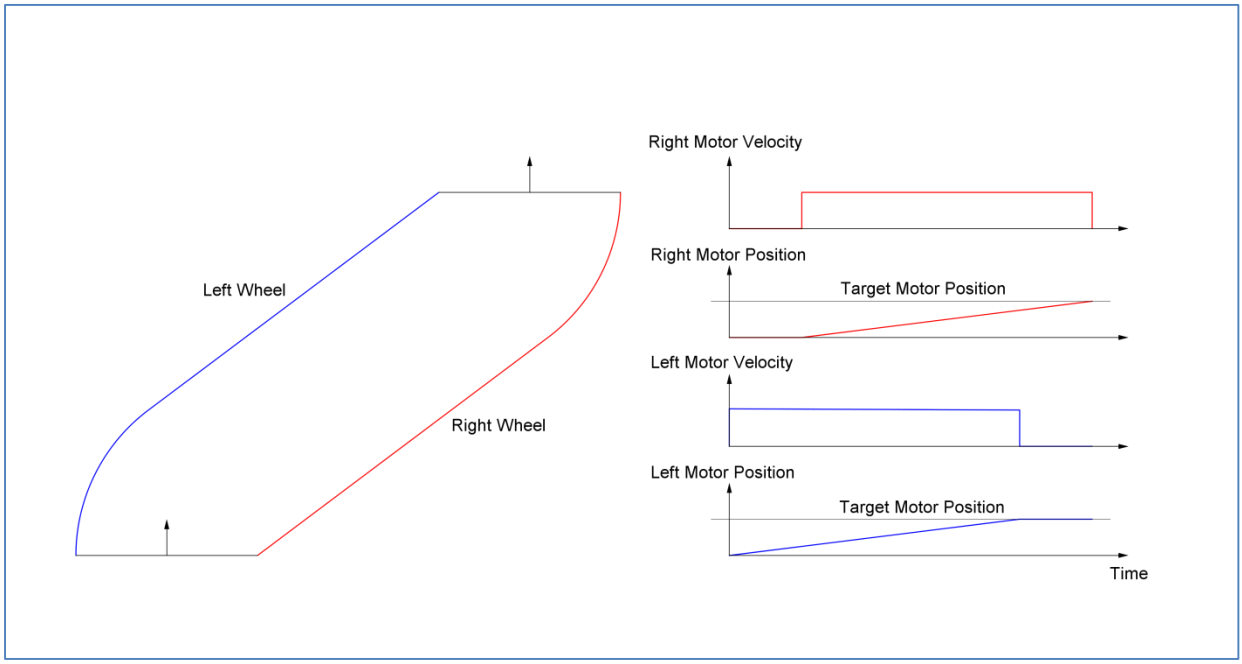

Figure 1: Example of a timing error while steering

An example of one of these errors can be seen in Figure 1. In this case, although both motors moved the same distance at the same speed before they stopped, the left motor moved slightly earlier. This causes the robot to move diagonally instead of straight forward. Although Figure 1 is only meant to give an idea about these kinds of problems and the error is drawn exaggeratedly, the issue occurs even if it is nearly unnoticeable. However, even if only marginal, errors like that cumulate and cause great discrepancies between the intended route and the real route, especially since the PID controllers used to control the motor velocities in the KIPR Link have a rather slow update speed.

To avoid the mentioned issues, our team developed a system called Robot Positioning System which will be described in the following chapter.

3. The Robot Positioning System

The RPS is a system devoted to taking care of all navigation related needs of a robot. The main tasks are steering and giving feedback about the current position. In general, the purpose of this system is allowing the user or programmer of the robot to simply specify a target

location and having the machine do all the minor tasks like turning towards the target, driving there, adjusting the course to errors and whatsoever. By doing so, it allows the user to enter a specified location either in reference to a point of origin of a coordinate system or in reference to the robot's current position. The following chapters are going to explain the separate systems that are necessary for the RPS and how they were implemented.

3.1. Utilizing the EMF feedback of Black Gear Motors

The Black Gear Motors included in the Botball electric set are standard full rotation servo motors that use DC-motors and a gear reduction to generate a rotatory force. These DC-motors are controlled by a controller IC that drives an H-bridge, with a pulse width modulation, which controls the current supplied to the motor.

Due to the electrical behavior of the anchor of a DC-motor, which can be expressed by a series of a resistor, a coil and a voltage source which represents the Electro Motive Force feedback of a motor, the measurement of the motor's speed is possible without the use of additional sensors like encoders or the like. This can be done by briefly suspending the PWM cycle which feeds the DC-motor with a voltage that would drive a current in the anchor of the motor. By doing so, the magnetic field in the coil diminishes and the voltage measured at the connectors of the DC-motor becomes equal to the voltage of the voltage source representing the EMF feedback of the motor. The EMF is the force, which describes the voltage induction into the electrical circuit caused by the rotation of the DC-motor. Due to the induction law this voltage is proportional to the rotation speed multiplied with the motor constant of the motor. Using this mechanism, the motor controller IC measures the speed of the motor and calculates the so-called motor position counter by integrating that rotation speed. The motor position counter is then used by the *get_motor_position_counter* and the *move_at_velocity* PID controller.

Normally, users would use the *move_at_velocity* method to move the two wheels of a robot with a desired speed. Because this method bears the problems described in chapter 2 RPS uses a different controller concept.

3.1.1. Calculation of the differential movement

The key feature of RPS is the calculation of the differential movement. This differential movement is the distance and angle which has been driven in a timeframe.

Contrary to expectations, the calculation behind the RPS is pretty simple and can be done in no more than 5 lines of code. There are multiple ways that all result in the x, y and angular movement but one method is exceptionally simple.

By assuming that the velocity has not changed in the sampling timeframe, which is a legitimate assumption, because the processing speed of the KIPR Link is high enough to do the calculations that are necessary in a short timeframe, it is possible to define the movement as an arc, as it can be seen in the left part of Figure 2. Using this knowledge it is possible to calculate the direction as well as the distance of the movement within the timeframe by linking the two differential motor position counters with simple mathematics.

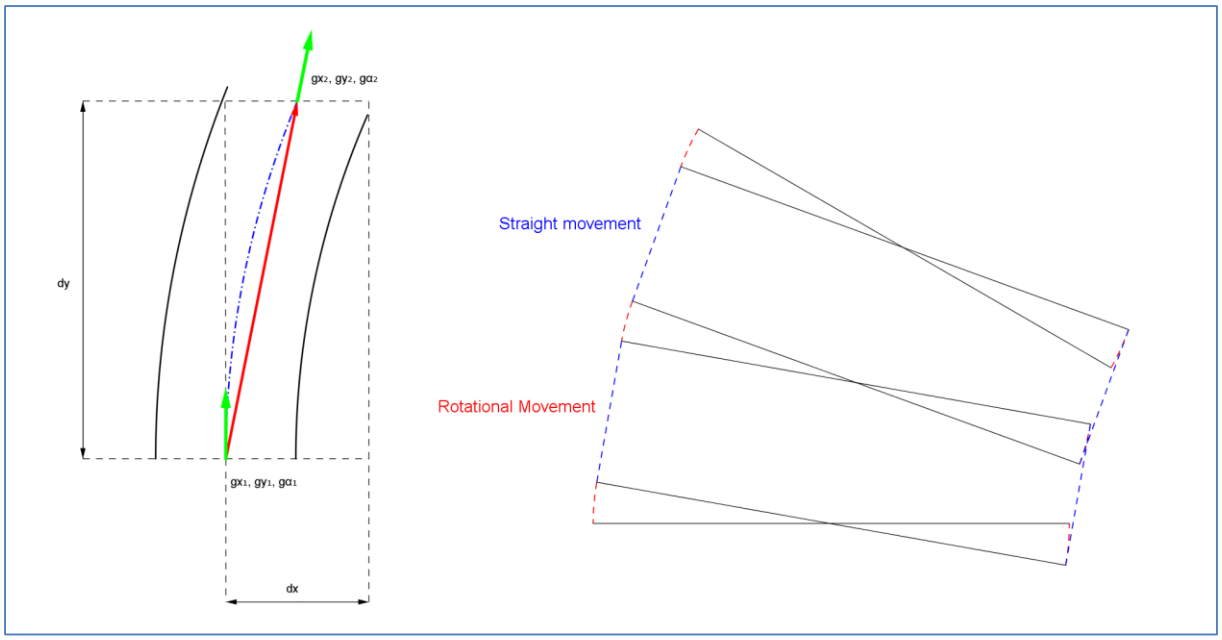

Figure 2: Visualization of the calculation

The main concept of this calculation is splitting the movement into separate rotational and forward movements. By quantifying the arc motion into tiny rotations and tiny forward movements like in the right part of Figure 2 it is possible to say that total movement consists of a rotation, which is the sum of all the tiny little quantified rotations, and forward movements that are executed at the same time. After sampling the motor position counters of a timeframe and subtracting the motor position counters before that timeframe the rotational movement can be easily calculated by simply taking twice the difference of the motor position counters of each motor. Once the rotational component is known the forward distance is also easily calculated and by using some simple geometry the value of the x, y and angular movement likewise.

3.2. Integrated motion

Using the calculated values from chapter 3.1.1 it is possible to integrate all the differential movements and thus gain information about the current position in reference to a start position or a virtual coordinate system's origin. The RPS provides this information in three values: the x-position, the y-position and the direction in which the robot is heading in degrees. These values are all in reference to a virtual Cartesian grid which is meant to have the positive x-axis pointing east, the positive y-axis pointing north, the point with the coordinates 0,0 in the southwest corner of the game board and all counterclockwise angles counted positive like it is defined in common geometry.

3.3. Implementation

Due to the fact that it is necessary to continuously keep calculating the differential movement and adding it to the current position, RPS uses a separate thread besides the main program thread to do these tasks. This thread is initialized at the beginning of the program and reads the motor position counter every 10 milliseconds. This 10 millisecond period was defined by pure arbitrariness and can be changed at will. However, this period should neither be chosen

too long, because that would increase the error caused by unsteady velocity during the motor position counter sampling timeframe, nor too short, since that would cause unnecessary loss of processor resources and a type of aliasing error which would occur by oversampling the motor position counter.

3.4. Utilization of the position data and movement algorithms

The position data gained from the RPS can be used in many ways. It gives the program feedback about the moved distance as well as error caused by friction and other influences.

The main usage of this position data is to control the movement of the robot. Due to the fact that the program has access to this data, it is possible to remove all controllers for the separate motor speeds, like the built in PID controller used for *move_at_velocity*, and control the power of the motors to steer into the desired direction. This workaround allows the robot to move at a high velocity without the necessity of having accurate motor speeds. Therefore, it is possible to move a robot using the maximum power of the motors without creating great errors since errors are also tracked by the calculation of the movement. Our team uses a simple algorithm which follows a virtual line between the robot's current position and the target position. Since this algorithm is not bound to any execution times due to the fact that that the robot drives until it reaches its target position, it is possible to simply implement acceleration and deceleration ramps.

Another handy application of the position data is the visualization of the robots position. Using the new graphics methods provided by KIPR, it is possible to draw a game board as well as a simple model of the robot on its current position. This drawing makes it possible to try out algorithms without the necessity of a real game board, since programmers can use the displayed robot position to check if the code is working correctly or not.

3.5. Achievements, advantages, disadvantages, issues and flaws

Using the RPS, we managed to create a steering method which allows the robot to execute extremely long movements without the necessity of realigning the robot at all, whether they are sensor based, of unknown length or include other unknown factors. Tests showed that driving a circle, a triangle or a square 20 to 30 times only caused a collective error of a few centimeters and a few degrees. The robot can even hit an obstacle and get stuck somewhere without losing track of its position. The RPS allows the usage of transmissions with a gear ratio of 3 and above which is simply impossible to control with the normal PID controllers used by *move_at_velocity* due to the great errors occurring during movements.

A great flaw the system has is that it is necessary to have the motors literally linked to the ground. The whole calculation is based on the assumption, that the distance that the motor moved is directly linked to the distance the robot moved on its wheels. Therefore, if the wheels slip even in the slightest, the whole system is useless. This creates many requirements to the hardware of the robot.

Another source of great problems are the programmed constants. The calculation requires the input of certain constants which are used to translate motor ticks into millimeters and degrees.

These constants need to be calibrated manually by using reference movements like 360° turns and 10 meter straight driving.

4. Conclusion and further developments

In its current state, the RPS is already useable and opens up many possibilities for the usage of a great variety of algorithms. Nevertheless there are still flaws and issues in the RPS. Cumulative error caused by the calculation as well as slipping wheels reduce the long-term accuracy and thus make it practically impossible to move extensive distances without the necessity of realigning the robot by using walls or whatsoever, although movements like these far exceed the scope of BotBall. There are still abstract constants that need to be defined and calibrated by the user after a great deal of manual testing which make user-friendliness hardly existent. Due to reasons like these there are still many improvements to make. There are two major additions which are planned by us:

One of the further plans is to write a program which automatically drives a sample movement and then asks the user to input the error. Using this method, the program would be able to calculate the corrected constant.

The second idea is the implementation of additional sensors. For example using the accelero- and gyrometers built in the KIPR Link it would be possible to detect collisions as well as other external forces. By gaining this information it would be possible to make the robot try to get back on its path after crashing into another robot. Another planned sensor implementation would be the Asus Xtion depth sensor, which could be used for obstacle detection. After detecting the obstacles, the program would use a path-finding algorithm to avoid colliding with them. This is already in work and partly functional.

5. Abbreviation table

The following abbreviations were used in this paper:

KIPR	KISS Institute for Practical Robotics
EMF	Electro Motive Force
PID	Proportional, Integral, Derivative
RPS	Robot Positioning System
DC	Direct Current
PWM	Pulse Width Modulation
IC	Integrated Circuit

6. References

[1] Bowditch, N. (2002). *The American Practical Navigator.*