Software implementation

Software usage for sensor, motor and camera optimisation

Sarah Breit, Julia Pöschl, Kacper Urbaniec, Barbara Wiedermann
TGM, Department of Information Technology
Vienna, Austria

Abstract

The following paper takes a look at the development and progress made so far. It features our problems and how we solved them by writing software programs to optimize and further enhance sensor, motor and camera usage. Our goal was to write many small code snippets with effective results.

***Keywords**: orientation, sensor usage, driving straight ahead, camera usage*

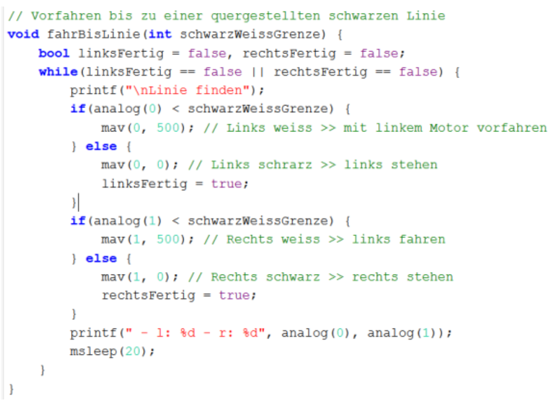
Introduction

Because we are all second year students in the IT department of the TGM we decided to choose the software related topic for our paper. We wrote a lot of easy and small functions which then helped us to write a bigger program much more easily. We also tried to even out some "hardware problems" with the software, for example, that two motors never have the same power.

Orientation

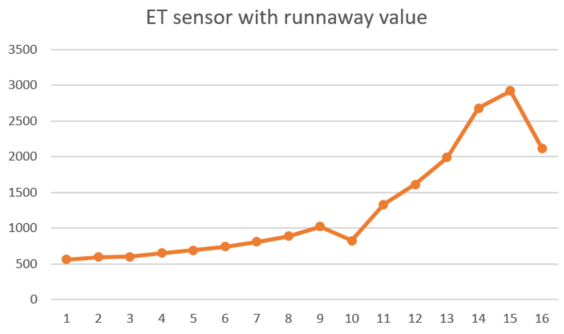
On this year's game table there are many black lines which are mainly there to differentiate the zones. However, many lines cannot be followed by the robot. So these lines are only used by the robot as orientation lines between the different zones on the game table.

We therefore mounted the two reflective sensors on the front of the robot (one on the right and one on the left) and wrote a function which detects the black line. The robot drives forward so long as both sensors are not on the black line. For example, when the right sensor sees black before the left, the right motor stops and the left motor keeps going until it is on the line too. This lets the robot stand exactly in a 90-degree angle to the line.



```c
// Vorfahren bis zu einer quergestellten schwarzen Linie
void fahrBisLinie(int schwarzWeissGrenze) {
    bool linksFertig = false, rechtsFertig = false;
    while(linksFertig == false || rechtsFertig == false) {
        printf("\nLinie finden");
        if(analog(0) < schwarzWeissGrenze) {
            mav(0, 500); // Links weiss >> mit linkem Motor vorfahren
        } else {
            mav(0, 0); // Links schrarz >> links stehen
            linksFertig = true;
        }
        if(analog(1) < schwarzWeissGrenze) {
            mav(1, 500); // Rechts weiss >> links fahren
        } else {
            mav(1, 0); // Rechts schwarz >> rechts stehen
            rechtsFertig = true;
        }
        printf(" - l: %d - r: %d", analog(0), analog(1));
        msleep(20);
    }
}
```

The same system can be used with two touch sensors driving against the PE pipes or two ET sensors.
Every time we used an analog sensor there were some runaway values. To solve this problem, we wrote a function which calculates the middle value of 10 values measured immediately after each other.

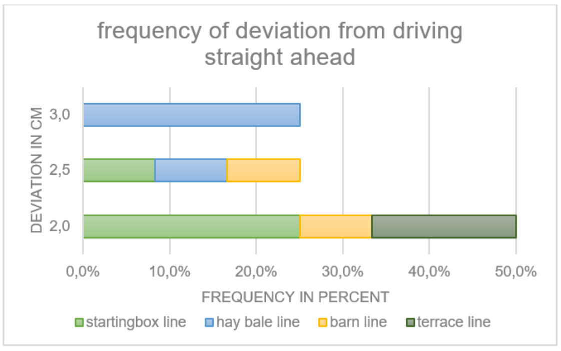


We also used this function for the data using the camera. (One time for the ET sensor and one time for the function "get_object_count (int chanel)")
If we wouldn't use this function the program would probably fail each time there is a runaway value. For example, if the ET sensor measures a small value

although the robot isn’t anywhere near to the object.

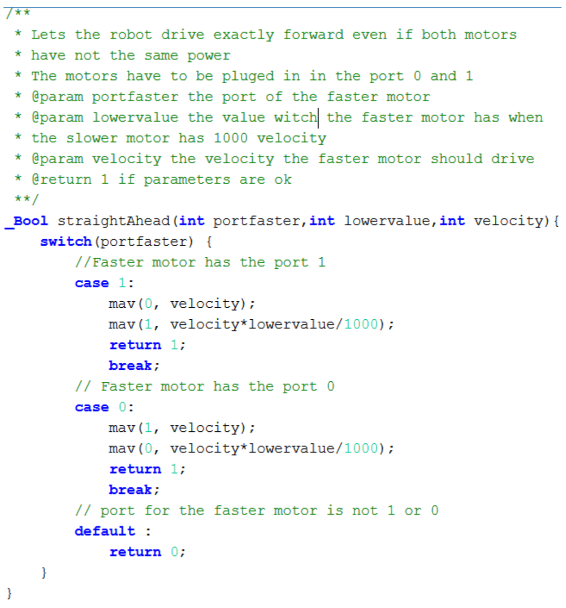
Problems with driving straight ahead

Problem with “move_at_velocity (int motor, int velocity)”
The problem with mav is that when two motors do not have the same power the robot does not drive straight ahead. Instead it drives a curve. Although the velocity is theoretically the same, the problem gets bigger the further the robot goes.



In this graph, you can see how often we measured the rounded up distance to the various lines on the game table. Our robot deviates about 2 to 3 cm to the right from the lines. This value is nearly constant. Because the robot had more often a deviation of 2 cm, the average is nearer to 2 than to 3. This shows that our robot drives to the right although both motors drive with the same velocity. So the power of the right motor is probably less than that of the left one.

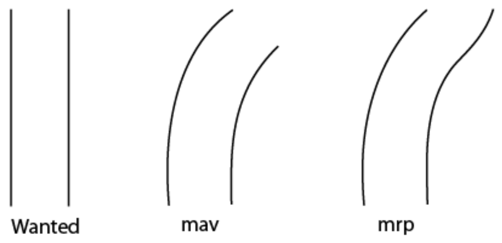
Our solution for driving straight:



```
/**
 * Lets the robot drive exactly forward even if both motors
 * have not the same power
 * The motors have to be pluged in in the port 0 and 1
 * @param portfaster the port of the faster motor
 * @param lowervalue the value witch the faster motor has when
 * the slower motor has 1000 velocity
 * @param velocity the velocity the faster motor should drive
 * @return 1 if parameters are ok
 **/
_Bool straightAhead(int portfaster,int lowervalue,int velocity){
    switch(portfaster) {
        //Faster motor has the port 1
        case 1:
            mav(0, velocity);
            mav(1, velocity*lowervalue/1000);
            return 1;
            break;
        // Faster motor has the port 0
        case 0:
            mav(1, velocity);
            mav(0, velocity*lowervalue/1000);
            return 1;
            break;
        // port for the faster motor is not 1 or 0
        default :
            return 0;
    }
}
```

Problem with “move_relative_position(int motor, int speed, int delta_pos)”

With mrp both motors drive the same amount of ticks but not in the same time.
Because of the different powers of the motors the robot at first drives the same curve as with mav but then the faster motor stops and the slower one continues to drive. This causes the robot to drive an “S” formed curve.



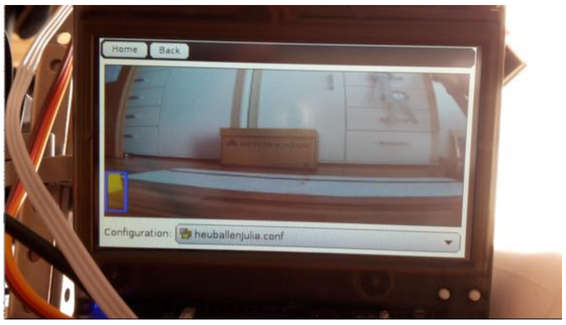
Because of that we wrote a function which modifies the velocity even though there are ticks. (just like we did with mav) It additionally also adjusts the amount of ticks.

Finding objects on the game table

The following paragraphs refer to finding the hay bales on the game table since it is a task this year. Because they stand in the middle of a line, there is almost no way to find them with distance and reflective sensors. So, we had to find an alternative. Our solution was to use the camera to detect yellow objects.

Because the camera in our kit did not work we used an action camera just for programming and testing until we are able to get another one. The only problem with this camera was that the width of the pictures was not the same as of the camera provided in the kit. So, we did not know the exact width of the pictures.

We then tried to find out the width with a program written by ourselves. Our program is very simple- we used the function "get_object_center_column(int channel, int object)" which returns the position of the middle of the object. Then we only had to position an object on the right edge of the camera view. We did this by using the camera test mode. For example, it returned us the value 311. But this is not the actual width of the picture. This is because the function returns the centre of the object. Because of that we put our hay bale on the left side of the camera view and ran our program again. Then the only thing we had to do was to take the value that we measured when the haybale was on the left and add it to the value we measured when the haybale was on the right. Now we also knew how wide a hay bale is.



Configuration: heuballenjulia.conf

Which hay bale should the robot take?

Because there are three hay bales next to each other we had to write an algorithm which chooses one. Typically, the function from kipr selects the objects based on their size[1]. This comes in handy very often but because these hay bales are the same size it causes a problem. The robot could not handle where to drive, the program could not decide which of the hay bales is the biggest.

To avoid this we measured the position of all yellow objects, and then chose the one which is more on the left than the others. Additionally we had to consider that the number of hay bales is going to shrink because we stack them in the barn area. We used the "get_object_count" function to count how many haybales are left on the black line, but set the limit to 3 to ignore other yellow smaller objects in the background.

```
cameraValue=0;
for(i = 0; i < 3; i++) {
    summ[i] = 0;
}
counter=0;
while(counter<10){
    camera_update();
    yellowObjects = get_object_count(0);
    if(yellowObjects > 3) {
        yellowObjects = 3;
    }
    if (yellowObjects > 0){
        for(i = 0; i < yellowObjects; i++ ) {
            summ[i]+= get_object_center_column(0,i);
        }

    }
    else{
        counter = counter-1;
        ao();
        printf("PANIK!!!!!\n");
        panicCounter++;
    }
    counter = counter+1;
}
// Linkestes Objekt
cameraValue = summ[0];
printf("[1]: %d - ", summ[0]);
for(i = 1; i < yellowObjects; i++) {
    printf("[%d]: %d - ", i, summ[i]);
    if(summ[i] < cameraValue) {
        cameraValue = summ[i];
    }
}
cameraValue = cameraValue/10;
```

We saved the values of the positions of all objects in an array and then searched for the lowest value with a simple searching algorithm. Then we used a simple if or else if the robot was to drive left or right.

References

[1] Documentation of the kipr libraries
http://192.168.125.1:8888/doc/group__camera.html#ga19c05235f84400070c5fcca43e0de185