Major Project Documentation

Stefan Klaus
stk4@aber.ac.uk
Aberystwyth, Wales

March 24, 2014

Contents

1 Background and Design **3**
1.1 What the Project is about . . . 3
1.2 Background . . . 3
1.2.1 SLAM - Simultaneous Localization And Mapping . . . 3
1.2.2 Deployment . . . 4
1.2.3 Communication . . . 4
1.3 Design . . . 5
1.3.1 Environment Design . . . 5
1.3.2 Mapping and Swarm Size . . . 6
1.3.3 Deployment . . . 6

2 Early Development **7**
2.1 Webots Tutorials . . . 7
2.2 First Program Iterations . . . 7

3 Mid stage Development **8**
3.1 Advanced tutorials . . . 8
3.2 Movement . . . 8
3.2.1 Computation of Odometry data . . . 8
3.2.2 Moving Forward . . . 10
3.2.3 Turn a given angle . . . 11

List of Figures

1 An example of the communication link . . . 5
2 Environment Design . . . 6
3 Movement pattern example . . . 7

1 Background and Design

1.1 What the Project is about

This project is about the map building using swarm intelligence.
The aspects of the project include SLAM(Simultaneous Localization And Mapping) as well as communication between a swarm of robots.
This document will disuse aspects such as robot localisation, deployment of the swarm, communication between robots and environment mapping.
My work is based on the research themes in these areas and I am using the robot simulator software Webots™. The mobile robot platform I am using is a virtual representation of the E-Puck robot.
The Programming will be done in C, using the Webots™ API.

One possible scenario for the program I am developing would be the mapping of an environment that would have become partially unknown after e.g. an nature catastrophe. I specify it this way since the team which would deploy this program on their robot swarm would most likely know the estimated size of the target environment, this is important in order to find an appropriate size for the robot swarm.
In such an scenario it is not always possible to achieve GPS coverage so the program should at its best function without the need for GPS localisation, so too find a localisation method which does not requires it is key. Another important features of the robot are obviously distance sensors, the E-Puck robot comes equipped with laser distance sensors, which I will use. Other features cover communication possibilities between robots, motor encoders which are needed for moving the robots in a controlled pattern and odometry calculations which will improve the movement and mapping significantly.

While it is true that the E-puck robot platform is most likely not suitable for such a scenario I use it because of the time limitation for this project and its good integration inside the Webots™ simulator.

1.2 Background

1.2.1 SLAM - Simultaneous Localization And Mapping

The SLAM problem is a current research topic which is based on different localisation algorithms and using a range of different sensor to effectively map an target area. A lot of different approaches have been done and many research papers have been written, the one I am basing my project on being a paper about a SLAM solution designed for autonomous vehicles [5].
While the research area of this paper is based on a much larger scale and only using 1 vehicle in comparison to flock, it does still give me an insight upon the SLAM problem.
E.g. the problem with localisation in an dynamic environment, the paper tackles this problem by using global reference points and a millimetre wave radar, however for my project I do use an static environment and simple laser range finders. So this paper is only used a reference to the localisation problem, especially the idea of using "global" reference points for the created map.
As the test environment and the sensors available for the E-puck sensors are limited I will assume that the starting location of the robots is known.

Another paper I read about this problem used an approach much more similar to my own project, by using different mobile robots which have no GPS access and simply use 2D laser range finders. However the approach described in this paper was based around the mapping of one "lead" robot and the traversing the same map again with a second robot using the map generated by the first for localisation purposes.
The second robot would then scan the target area again and refine the already generated map though using the (now stationary) first robot as an reference point. Since my project is planed about using multiple robot which scan the area at the same time it does still gives some information and idea about map sharing and improving.
Since my swarm will traverse the area at the same time it would be needed to share the map in real time and know the robot location in comparison to the first global referring point. By implementing this I could rescan an area if a robot traverse an area another robot already scanned and refine the the final map by this. I will however look into real time sharing between different robots and can not say at this point if I will implement this in my final solution, it is however a interesting thought.

1.2.2 Deployment

The deployment strategy is an important part of my project as it defines how effective the swarm will cover the target area which will define how long it will take to scan and map the whole area. Another important aspect is how many robots can the swarm hold and effectively deploy using the current deployment strategy.
One research paper I found proposed an solution of a communication network where the comm nodes keep track of the robots positions and guide them in directions which have not been explored in the last time period [3]. The paper uses a solution which is based on small comm nodes deployed by the robot, to make the solution fitting for my project I would have to define some of my E-pucks as communication nodes which remain on a fast position and guide the "scout" E-pucks based on area which have been least visited by the other robots.
This is certainly an possible solution however it could be considered a waste of robots in an small environment. Since I am using an simulator there is no communication range problem, but as my indentions are to make it as close to a possible real world application as possible I must still consider this. That is why I am going to implement an maximum communication range for the robots however more about that can be found inside the "Communication" section. It is however an aspect to which I will come back later in my testing.

Another paper proposed an solution which is based on an Nearest-Neighbour algorithm, meaning every robot must have always a minimum of "N" other robots inside its communication range [9]. In this solution to robots would emit signals to other robots which would manoeuvre the robots away from each other until only "N" robots remain inside the robots communication range.
This solution would cover a large area with the swarm fairly fast and also be adaptable for a swarm of any size, however a solution of moving the whole swarm in on direction would be needed in order to cover the whole target area, assuming the robot swarm is not big enough to cover it once completely deployed. This would therefore need both a lead robot which decides the movement of the swarm and communication robots which would always have at least 1 link to another comm robot in order to have a communication line back to the lead robot.
This is one of two 2 deployment strategies I will try to implement during the testing period and try to find out which one would be more fitting for my project.

1.2.3 Communication

While it is possible for me to transfer information easily between robots since I am using a simulator I am still trying implement it as close to a realistic scenario as possible, meaning that the communication range for the robots is limited. In an realistic scenario every robot would have to send the acquired map back to the static start point/lead robot so that an overall map of the environment can be created. Since the communication range for such small robots is limited and can be even further obstructed through obstacles like walls it is important to designated some robots as communication nodes. Such comm nodes would than remain stationary and link the "scout" robots, which do the exploration, back to do the lead robot.
Obviously the most effective way to do this is by implementing different behaviour patterns for scouts or comm robots, and implement a decision model which allows the robot to change between either pattern as the needs of the swarm change. E.g. in the start of the exploration no comm robots will be needed as the robots would most likely be inside the comm range of the lead robot, though this may change if the swarm is big and spread out enough.

To surpass the problems of obstacles obstructing the communication the comm robots would need to position them self on logical places i.e. in order to scan a room it would be important that a comm robot places it self inside, or close to, the doorway so that others can explore the room and still communicated back to the rest of the swarm. The robot would need to stay inside the doorway as signals can not always travel through walls and the energy reserves of mobile robots are limited so they most likely can not send high power signals.
I have yet to decide how I am going to implement the communication part of in the project as I am not sure if the E-puck models implemented inside the simulator are able to send signals to other robots. While I know that it is able to transfer signals through the E-puck's laser sensors I do not know if I will implement it as this is fairly difficult and time consuming. I will come back to this should I have enough time available at the end of the project or if no other alternative can be found.

The theory of what I am going to implement uses a defined maximum communication range for the robots and a grouping strategy which specifies that each scout robot need to stay in contact which at least 1 comm robot while the comm robots always need at least 1 other comm robot inside their communication range. If implemented correctly the comm robots would on this way create a communication link back to the lead robot/starting location which the scout robots can use to transfer all new information back.

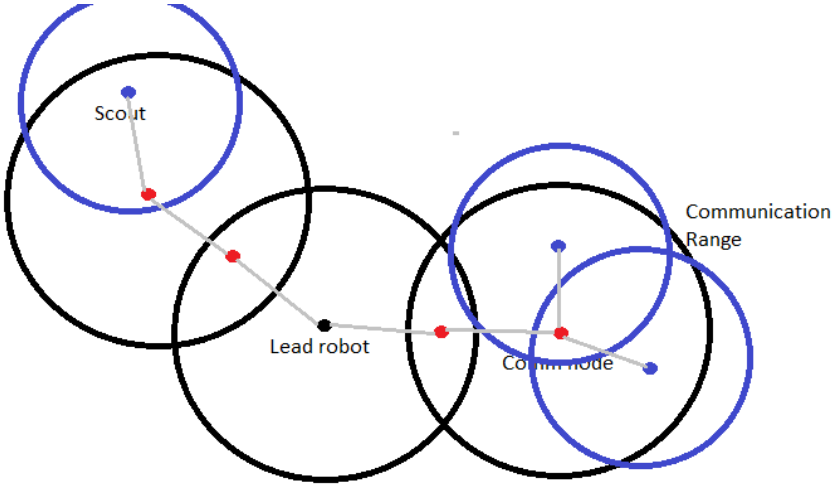

Figure 1: An example of the communication link

Figure 1 shows one possible example of the communication link, where the lead robot/ or in some cases a stationary uplink point is in the center and the communication robots(in red) placed in such positions that their comm range overlaps the comm range of other robots and the lead robot.
This configuration allows the scouts(in blue) to move and explore anything inside the communication range of the different comm robots. When the robots at the right side of the figure would now try to move outside the comm range one of them would have to change their behaviour pattern to "communication mode" at the outer range of the other comm nodes range while the last remaining scout continuous exploring in this direction.
This example shows that it is important to have a swarm of a suitable size for an environment to be able to cover at much area with the robots at hand and for cases in which this is not possible to be able to move the whole swarm in one unified direction to explore unmapped locations. This is however only doable when there is a lead robot since a stationary comm/uplink point is by definition, stationary.

1.3 Design

In this section I will describe my design as of the date of the project outline description.
This is the design I will follow and try to implement, however this is more of a guideline rather than exact plan since I have to figure out what the API and simulator are able to do and what I am able to implement inside the given time.

1.3.1 Environment Design

The final environment which I will create for this project will consist of a large starting area, with a corridor adjacent to it. The corridor will hold 1 smaller room to either side, 1 of them with an obstacle inside which will have to be traversed and mapped. The size of the corridor should not be to big so that it is still a challenge to move through it however it should rather not be to small to make movement through it too problematic.
A to small size could also make the corridor not distinctive enough from the rest of the wall so the E-Puck could have problem registering and moving into it.

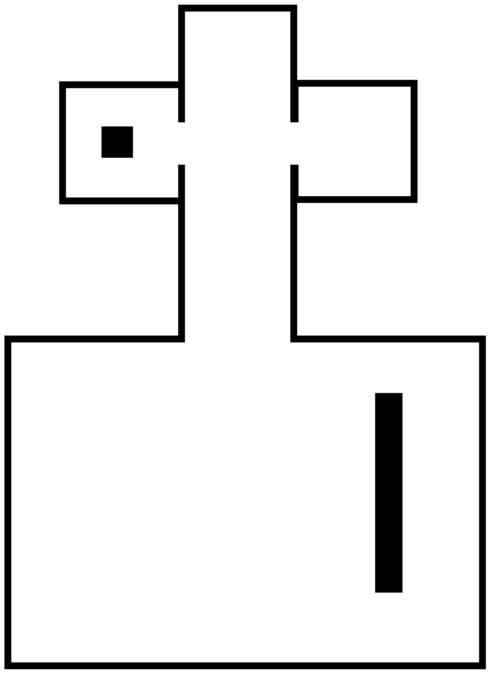

Figure 2: Environment Design

1.3.2 Mapping and Swarm Size

For mapping I will use a occupancy grid, and the occupancy will be acquired by the E-Puck's laser sensors. I will yet have to decide on a resolution, for the grid.
I will use a swarm of 5 E-Puck robots, of which 1 will remain stationary and only function as the "Uplink point" to which all robot send the acquired data. The stationary robot will also be used as reference point for the localisation method.

1.3.3 Deployment

I am of this moment still undecided for which deployment strategy will be implemented.
There are 2 deployment strategies which I decided for based on the background research I have done.

One of them is based on a nearest neighbour approach in which the robots will emit signals to each other which will cause them to drive away from each other until a previous maximum communication distance has been reached. This relationship between robots could be described similar to magnets which can repulse and pull in each other in order to hold a given maximum distance and thereby cover the largest possible area
The movement of the robots would be a random walk with included obstacle avoidance. One possibility could be to add a wall following functionality to it as well. However this would require some form of higher functions which to keep the robot from being stuck inside a loop. These higher functions could work by keeping track of the robots position on the map and react once the robot starts traversing the same area again, in this case the algorithm could force to robot back into a random walk.

The other deployment strategy would implement a more controlled movement pattern. This pattern would move the robots inside a rectangular pattern which would implement a function to move around obstacles on the way before moving back into the original pattern. While this reason is more controlled I am not sure which one will turn out to be more effective, that it why I will implement both inside a testing phase and will then decide which of them I will use.

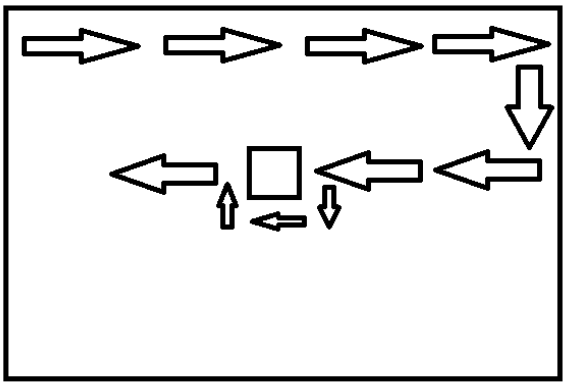

Figure 3: Movement pattern example

2 Early Development

After the first weeks where background reading were done and designing a plan for the final program, I started working with the Webots™ simulator.
More information about the different stages can be found in the following sub-sections.

2.1 Webots Tutorials

In the beginning I was following different tutorials and demo programs included in the Webots installation. This helped me to see how the simulator works, what it can do and what possibilities the Webots API gives me.
The tutorials I used were provided by the Webots wikibooks site [1].
Starting off with simple tutorials and movement and sensor reading I was soon able to implement simple programs which were based on simple forwards movement with obstacle avoidance functions, based on the proximity sensor values. After I had done some of the more "advanced"(in comparison to what I had done to this point) tutorials which involved e.g.: reading the robot encoders I took a look at example programs of more advanced types of movement as well as SLAM examples.

2.2 First Program Iterations

Taking my experience of the tutorials I moved on to actually implementing the first prototype of my program.
The first prototype was still heavily based on a Webots example program called "Intermediate Lawn Mower" which involved a simple movement pattern based on a basic finite state machine(FSM).
My first attempt was based recreating the FSM to get the same basic movement capabilities as the the program it was based upon and than changing it to achieve the level of movement control I needed.

In the following iteration I were trying to implement the functionality of moving a given distance based on resetting the stepper motors encoders and moving the robot until it reaches a given encoder value. I quickly realised that this would not work with the way the FSM was currently implemented so I decided to move the functionality of the FSM, which was currently based on a simple switch statement, to a set of separate functions. The idea was that this would give me the freedom I needed to be able to call a method, i.e. move_forward(), and keep it running until a predefined encoder value has been reached. However this did not work based on the overall way my program was designed at this point. It was designed around an switch statement which based its decisions on the proximity sensors of the E-Puck and then setting the movement speed for the robot motors. As it was based on this calling separate functions to move and turn did not work, which resolved in problems around my current idea of using the stepper motor encoders.

As I needed to be able to move a certain distance in order to effectively implement the rectangular

[1] `http://en.wikibooks.org/wiki/Cyberbotics%27_Robot_Curriculum`

movement pattern, I tried various approaches which were all based around the FSM. I realised soon that I was not getting anywhere which this way of thinking so I decided to come back to this problem later and went to another problem, turning a given number of degrees.
As the simulator simulates friction between the robot wheels and the environment the turning function, as it was in it current state, was far to inaccurate to be usable.
It was based on a very basic odometry calculation, taking the encoder values and calculating the turning distance based on the wheel diameter and axle length of the robot. While this is not a bad approach it had no procedures of slowing down the movement as it got closer to the target state so overshooting the wanted position or rotating far to less if the motor speed would have been set to low.
I tried countering the problem by alternating the values it used for the odometry calculations, and while I was able to come close a solution it was far from accurate. Another problem with this solution was that it only worked for 90 degree rotations based on the modified values, meaning all rotations to another point were impossible without modifying the values to fit the target bearing. Which was counter productive as it would be best to have 1 function able to turn the robot to any wanted bearing.

3 Mid stage Development

After the problems I encountered during my first program iterations, I quickly realised that my way of thinking and understanding of the API was flawed. A lot of the problems I encountered were the result of insufficient programming skills for what I was trying to do combined with bad understanding of how odometry worked, and how I could use it to control the robot.
Seeing these kinds of problems I decided to take a step back as I am clearly "moving in circles" and encountering the same kind of problems over and over again with my current approach.
I decided to take once more a look at the provided example programs and to study their odometry functions in order to gain some new insight into odometry calculations, and find a new approach based on that.

3.1 Advanced tutorials

The example programs provided together with the simulator are categorised after the estimate level of knowledge needed to complete them. Theses categories are: Beginner, Novice, Intermediate and Advanced. I already looked at the advanced programs before as they feature a couple of examples on odometry and slam, however I did not understand it well enough at the time.
Looking closer at the odometry functions provided I managed to understand a bit better how the advanced programs worked and how they calculated the movement of the robot.

I also took a look at the code and notes I took during my Robotics module on the second year, while the API worked different for the Player/Stage environment the idea behind rotation and movement was still the same and had only to be applied using the Webots API.

3.2 Movement

After I have studied the odometry functions of the provided programs I decided to start a new approach to fix the movement of the robot.
This approach is based on a set of different functions, much smaller and more refined than my previous approach. This approach took a while to implement and test but the results were rather satisfactory. In this section I am going to describe the different aspects of my movement solution and describe how the most important functions work.
I will also post the code to each of the major functions as it will help with describing them.

3.2.1 Computation of Odometry data

The function calculate and return the odometry data works by saving a global buffer of current and previous encoder positions of the stepper motors. It then calculates how many steps(ticks) the each encoder has done by comparing the previous with the current encoder values. While theses values should remain vaguely the same, since there is friction in the environment small differences will happen, they will be completely different when the robot turns.

After having calculated the amount of steps taken by each motor it will calculate the distance each wheel has travelled by dividing the number steps by the encoder resolution timed with the wheel radius, where both values have been taken from an example program. It will then calculate the displacement from the center location by dividing the sum of the distances for the each wheel by 2. And calculate the rotation, theta, by dividing the difference of the distances between the motors with the size of the wheelbase. The wheelbase values has also been taken from another example program.
The new X and Y location of the robot can be calculated from the value of the displacement from the center and the current rotation of the robot. It will then modify the global buffer for the encoder positions and return an array with the robot's current X and Y location and its rotation.

```
/**
Function to compute the robots current odometry data
this includes the x and y placement as well as the rotation theta
*/
double* compute_odometry_data(){
        static double dOdometryData[3];
        double *point_dEncPos;

        double dNumTicksLeft = 0.0f;
        double dNumTicksRight = 0.0f;
        double dLeftDist = 0.0f;
        double dRightDist = 0.0f;
        double dDispCenter = 0.0f;
        static double dTheta = 0.0f;
        static double dPosX = 0.0f;
        static double dPosY = 0.0f;

        //get encoder position
        point_dEncPos = get_encoder_positions();

        /* calculate the distance travled for the wheels
        (number of ticks done since the previous encoder position
        was recoreded)*/
        dNumTicksLeft = point_dEncPos[0] - dPrevEncPos[0];
        dNumTicksRight = point_dEncPos[1] - dPrevEncPos[1];

        //convert the number of encoder ticks to a real distance
        dLeftDist = dNumTicksLeft / ENCODER_RESOLUTION * WHEEL_RADIUS;
        dRightDist = dNumTicksRight / ENCODER_RESOLUTION * WHEEL_RADIUS;

        //calculate the displacement distance from the center
        dDispCenter = (dLeftDist + dRightDist)/2.0f;

        //calculate the rotation theta of the robot
        dTheta += (dLeftDist - dRightDist)/(WHEELBASE);

        //calculate Cartesian coordinates (x, y) of the robot
        dPosX += dDispCenter * sin(dTheta);
        dPosY += dDispCenter * cos(dTheta);

        //update the previous encoder positions
        dPrevEncPos[0] = point_dEncPos[0];
        dPrevEncPos[1] = point_dEncPos[1];

        //write the calculate values into the buffer and return
        dOdometryData[0] = dPosX;
        dOdometryData[1] = dPosY;
        dOdometryData[2] = dTheta;

        return dOdometryData;
}
```

3.2.2 Moving Forward

The move_forward function takes 2 doubles as parameters, 1 being the speed with which the robot is ordered to move and the distance it should move.
The function first checks that neither of the parameters are 0 and then calculates the number of steps each motor has to drive(1 step being 1 step of the stepper motor) to reach its target position. This calculation is done by dividing the number of steps needed for a full wheel rotation, 1000 in this case, by the product of π times the wheel radius times 2 and multiplication this result with the distance defined in the parameter of the function. The steps needed for a full rotation have been taken from the E-Puck documentation and have been confirmed during on of the odometry tutorials.
It will then read the current encoder positions and calculate the stop position for each motor by the sum of the encoder values for each motor and the previously calculated encoder steps needed to reach the target area. It will then set the motor speed based on the speed defined in the parameters.

It then enters the control code which will stop the robot once it reaches it target, position.
This is controlled by comparing the current encoder positions with the calculated target encoder positions and updating them all the time. Once the robot reached a pre-defined minimum distance of 20 encoder steps it will slow down the movement to a minimum speed of 10 steps per second. This is down to prevent the robot from overshooting the target area should it move with to much speed. The optimal minimum distance and speed has been found experimentally, and both values give good results and also prevent the robot from undershooting. Once it has reached it target location it will stop the robot, and force the simulator to take a simulation step, effectively moving onward to the next command.

This function allows me to move forward and stop after the predefined distance within a minimal error space, which will always exist given the friction simulated inside the simulator. This method required a lot of testing in order to get right as first versions did not include the control statement which slowed down the robot after a minimum difference between the encoders and the target encoder value has been reached. So the robot used to overshoot the target.
After the control statement was implemented it still required some testing and calibration of the minimum difference and speed values in order to avoid over and undershooting. However the found values work well and the movement error has been reduced to minimum.

```
/**
Function to turn the robot a given angle with a given speed
*/
void turn_angle(double dAngle, double dSpeed){
	double dFactor = 0.0f;
	double dStepCount = 0.0f;
	double dStopPosLeft = 0.0f;
	double dStopPosRight = 0.0f;
	double *point_dEncPos;

	if((dAngle != 0.0f) && (dSpeed > 0.0f)){
		//calculate turn factor
		dFactor = fabs(360.0f/dAngle);

		//calculate the number of step counts for the rotations
		dStepCount = (INCREMENT_STEP * WHEELBASE)/(dFactor * WHEEL_RADIUS * 2.0f);

		point_dEncPos = get_encoder_positions();

		//turn right
		if(dAngle > 0){
			//calculate the target encoder positions
			dStopPosLeft = point_dEncPos[0] + dStepCount;
			dStopPosRight = point_dEncPos[1] - dStepCount;

			point_dOdometryData = compute_odometry_data();

			set_motor_speed(dSpeed, -dSpeed);
```

```
                while((point_dEncPos[0] < dStopPosLeft) && (point_dEncPos[1] >
                    dStopPosRight)){
                        //get odometry data
                        point_dOdometryData = compute_odometry_data();

                        //get wheel encoders
                        point_dEncPos = get_encoder_positions();

                        //slow down the closer the robot come to the destination, reduces
                            the error
                        if(fabs(dStopPosLeft - point_dEncPos[0]) <= MIN_DIST){
                                set_motor_speed(MIN_SPEED, -MIN_SPEED);
                        }
                }

        } else { // turn left ...
                dStopPosLeft = point_dEncPos[0] - dStepCount;
                dStopPosRight = point_dEncPos[1] + dStepCount;

                // turn left the robot ...
                set_motor_speed(-dSpeed, dSpeed);

                while( (point_dEncPos[0] > dStopPosLeft) &&(point_dEncPos[1] < dStopPosRight) ) {

                        point_dOdometryData = compute_odometry_data();
                        point_dEncPos = get_encoder_positions();
                        if( fabs(dStopPosLeft - point_dEncPos[0]) <= MIN_DIST ){
                                set_motor_speed(-MIN_SPEED, MIN_SPEED); }
                        }
                }
        }
        stop_robot();

        //update odometry data
        point_dOdometryData = compute_odometry_data();
        wb_robot_step(TIME_STEP);
}
```

3.2.3 Turn a given angle

The turn_angle function takes 2 doubles as parameters, one being the angle the robot will turn to the other the speed with which the robot will turn.
First the factor by which the robot will turn is calculated by dividing 360, the value of a full rotation, with the defined angle. Once the factor has been calculated it will then the number of steps the motor have to do until the target position is reached. This is done by dividing the product of the steps needed for a full wheel rotation, 1000, and the size of the wheelbase by the product of the calculated turning factor and 2 times the wheel radius. It will then use a function to return the current motor encoder positions. This function simply uses the Webots™ API and returns the values.

If the rotation angle defined as the function parameter is positive the robot will turn to the right. When the robot is turning to the right it will calculated the stopping positions of the encoders by adding the calculated step count to the left motor encoder and subtracting it from the right encoder. This will lead to the wheels turning against each other and will result in the robot turning on the spot rather than only moving 1 wheel to turn which would result in a displacement of the robot.
It will then update the global odometry buffer by calling the compute_odometry_data function and set the the speed of the motors using the given function parameter value. The right motor will receive a negated value so that it will turn backwards. It will then compare the left encoder positions and updated them all the time. Similar to how the forward movement function worked, it will detect when a given minimum difference between the current and target encoder values is reached and slow the robot down to a minimum speed.

If the rotation angle defined as the function parameter is negative the robot will turn to the left.
The only difference between turning left rather than to the right is that the calculations, obviously, are reversed. Meaning to calculated the stop positions of the motor encoders it will subtract the calculated step count from the current left encoder value and add the step count the the right encoder value, same switch of negation has been done where the motor speeds are set. The calculations of how long to turn and when to slow down are identical to how they work when turning right, only difference being that the the operators to which check how long to turn are different.
Once the target position has been reached, by either turning left or right, the robot stops and the global odometry buffer is updated. It will then force to the simulator to take a simulator step, effectively moving on to the next command.

This functions allows me to define the turn the robot by so many degrees as I need and it will turn there within an minimal error space. This error space exist because the simulator simulates friction between the robot wheels and the environment so 100% accurate movement will never happen.
There also existed the problem of over/undershooting with the turning however the values found during tests of the move_forward function turned out to also work well for the turning function. However one problem remains, since there never is going to be a perfect rotation the error value will add up over time, resulting in less and less accurate turns, overshooting the target rotation is going to be a real problem. I have at this point not yet a solution for this problem, however the function works well and is a great improvement to how it turning was implemented in previous iterations of the program.

```
/**
Function to turn the robot a given angle with a given speed
*/
void turn_angle(double dAngle, double dSpeed){
        double dFactor = 0.0f;
        double dStepCount = 0.0f;
        double dStopPosLeft = 0.0f;
        double dStopPosRight = 0.0f;
        double *point_dEncPos;

        if((dAngle != 0.0f) && (dSpeed > 0.0f)){
                //calculate turn factor
                dFactor = fabs(360.0f/dAngle);

                //calculate the number of step counts for the rotations
                dStepCount = (INCREMENT_STEP * WHEELBASE)/(dFactor * WHEEL_RADIUS * 2.0f);

                point_dEncPos = get_encoder_positions();

                //turn right
                if(dAngle > 0){
                        //calculate the target encoder positions
                        dStopPosLeft = point_dEncPos[0] + dStepCount;
                        dStopPosRight = point_dEncPos[1] - dStepCount;

                        point_dOdometryData = compute_odometry_data();

                        set_motor_speed(dSpeed, -dSpeed);

                        while((point_dEncPos[0] < dStopPosLeft) && (point_dEncPos[1] >
                            dStopPosRight)){
                           //get odometry data
                           point_dOdometryData = compute_odometry_data();

                           //get wheel encoders
                           point_dEncPos = get_encoder_positions();

                           //slow down the closer the robot come to the destination, reduces
                               the error
                           if(fabs(dStopPosLeft - point_dEncPos[0]) <= MIN_DIST){
                                   set_motor_speed(MIN_SPEED, -MIN_SPEED);
```

```
                        }
                }

        } else { // turn left ...
                dStopPosLeft = point_dEncPos[0] - dStepCount;
                dStopPosRight = point_dEncPos[1] + dStepCount;

                point_dOdometryData = compute_odometry_data();

                // turn left the robot ...
                set_motor_speed(-dSpeed, dSpeed);

                while((point_dEncPos[0] > dStopPosLeft) &&(point_dEncPos[1] < dStopPosRight)){

                        point_dOdometryData = compute_odometry_data();
                        point_dEncPos = get_encoder_positions();
                        if( fabs(dStopPosLeft - point_dEncPos[0]) <= MIN_DIST ){
                                set_motor_speed(-MIN_SPEED, MIN_SPEED); }
                        }
                }
        }
        stop_robot();

        //update odometry data
        point_dOdometryData = compute_odometry_data();
        wb_robot_step(TIME_STEP);
}
```

References

[1] M. Abolhasan, T. Wysocki, and E. Dutkiewicz, "A review of routing protocols for mobile ad hoc networks," Ad Hoc Networks, vol. 2, no. 1, pp. 1–22, Jan. 2004. [Online]. Available: http://dx.doi.org/10.1016/s1570-8705(03)00043-x

[2] N. Agmon, N. Hazon, and G. Kaminka, "Constructing spanning trees for efficient multi-robot coverage," in Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on, May 2006, pp. 1698–1703.

[3] M. Batalin and G. Sukhatme, "Coverage, Exploration, and Deployment by a Mobile Robot and Communication Network," in Information Processing in Sensor Networks, ser. Lecture Notes in Computer Science, F. Zhao and L. Guibas, Eds. Springer Berlin Heidelberg, 2003, vol. 2634, pp. 376–391. [Online]. Available: http://dx.doi.org/10.1007/3-540-36978-3_25

[4] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun, "Collaborative multi-robot exploration," in Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on, vol. 1. IEEE, 2000, pp. 476–481 vol.1. [Online]. Available: http://dx.doi.org/10.1109/robot.2000.844100

[5] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," Robotics and Automation, IEEE Transactions on, vol. 17, no. 3, pp. 229–241, June 2001. [Online]. Available: http://dx.doi.org/10.1109/70.938381

This paper proves that SLAM is possible and gives a solution using global referance points for map building. The solution in this paper however was used on only a single car rather than a flock and using a millimeter wave radar for localising different obstacles and referance points. This makes this solution not applicable for my project, however it gave me some ideas concerning referance points

[6] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (slam) problem," Robotics and Automation, IEEE Transactions on, vol. 17, no. 3, pp. 229–241, Jun 2001.

[7] D. Fox, W. Burgard, and S. Thrun, "Markov Localization for Mobile Robots in Dynamic Environments," in Journal of Artificial Intelligence Research, vol. 11, 1999, pp. 391–427. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.372

[8] Y. Mei, Y.-H. Lu, Y. C. Hu, and C. S. G. Lee, "Deployment of mobile robots with energy and timing constraints," Robotics, IEEE Transactions on, vol. 22, no. 3, pp. 507–522, June 2006. [Online]. Available: http://dx.doi.org/10.1109/tro.2006.875494

[9] S. Poduri and G. Sukhatme, "Constrained coverage for mobile sensor networks," in Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on, vol. 1. IEEE, Apr. 2004, pp. 165–171 Vol.1. [Online]. Available: http://dx.doi.org/10.1109/robot.2004.1307146

This paper gives a good solution to the flock deployment and communication problem using a nearest-neighbor approach. This approach is scalable for a flock of any size and is my base idea for the deployment I will implement in my program

[10] C. W. Reynolds, "Flocks, Herds and Schools: A Distributed Behavioral Model," in Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, ser. SIGGRAPH '87, vol. 21, no. 4. New York, NY, USA: ACM, July 1987, pp. 25–34. [Online]. Available: http://dx.doi.org/10.1145/37401.37406

While not entirely relevant to my problem it rises a few important points about flock behavior. These include collision avoidance between members of the flock and the reduced perceptance of members which hare moving in the middle of the flock. Collision avoidance between members will play a big role when trying to navigate the flock through a tight environment.

[11] K. Singh and K. Fujimura, "Map making by cooperating mobile robots," in Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on. IEEE, May 1993, pp. 254–259 vol.2. [Online]. Available: http://dx.doi.org/10.1109/robot.1993.292155

[12] S. Thrun, W. Burgard, and D. Fox, "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping," in Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on, vol. 1. IEEE, 2000, pp. 321–328 vol.1. [Online]. Available: http://dx.doi.org/10.1109/robot.2000.844077

Methods in this paper are based on real robots and use odometry data. Their approach uses sample data from the sensors and the odometry calculation. This sample data is then used in a gradient descent algorithm with each sample as a starting point. This guarantees that the global maximum likelihood can be found, and by that the best map. Their approach also works with the absence of odometry data, as long as the world has sufficient variation. Should the world be to feature less no accurate map will be generated. The map generated using the odometry data will always be more accurate. Though the title implements the usage of multiple robots the only documented approach to this is to rescan the map after a original map has been build by the "lead" robot. The second robot then localises it self on the map and improves the map, using the position of the first robot as a referance point for the already generated map. This appears to be highly accurate, however not entirely useful for my approach. Uses Monte Carlo localisation which is similar to Markov localisation.

[13] S. Thrun, "Learning Occupancy Grids with Forward Models," in In Proceedings of the Conference on Intelligent Robots and Systems 2001, 2001, pp. 1676–1681. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.6014

[14] S. Thrun, W. Burgard, D. Fox, H. Hexmoor, and M. Mataric, "A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots," in Machine Learning, 1998, pp. 29–53. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.1128

[15] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo Localization for Mobile Robots," 2001. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.8488

[16] H.-X. Yang and M. Tang, "Adaptive routing strategy on networks of mobile nodes," Physica A: Statistical Mechanics and its Applications, vol. 402, pp. 1–7, May 2014. [Online]. Available: http://dx.doi.org/10.1016/j.physa.2014.01.063