Odometry based Map Building

Final Report for CS39440 Major Project

Author: Stefan Klaus (stk4@aber.ac.uk)

Supervisor: Dr. Myra Wilson (mxw@aber.ac.uk)

21st April 2012

Version: 1.0 (Draft)

This report was submitted as partial fulfilment of a BSc degree in AI & Robotics, (GH76)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.

- I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.

- I have read the sections on unfair practice in the Students' Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.

- I understand and agree to abide by the University's regulations governing these issues.

Signature ....

Date ....

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature ....

Date ....

Acknowledgements

I am grateful to...

I'd like to thank...

Abstract

Include an abstract for your project. This should be no more than 300 words.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Background & Objectives

This section should discuss your preparation for the project, including background reading, your analysis of the problem and the process or method you have followed to help structure your work. It is likely that you will reuse part of your outline project specification, but at this point in the project you should have more to talk about.

Note:

- All of the sections and text in this example are for illustration purposes. The main Chapters are a good starting point, but the content and actual sections that you include are likely to be different.

- Look at the document on the Structure of the Final Report for additional guidance.

1.0.1   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 this project is based on is a paper about a SLAM solution designed for autonomous vehicles [4].
While the research area of this paper is based on a much larger scale, 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 this project will assume that the starting location of the robots is known.

Another paper which was read about this problem used an approach much more similar to this 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 this project is using single a robot for mapping purposes and multi robot usage would only be added if enough time is available, this paper was not inherently useful, however gave some useful insight for information sharing between robots or sensor stations as well as localisation of robots using a global reference point.

Since this project aims at real time localisation and mapping communication with a "uplink" point would be essential to achieve this in a real world setting. By using a similar implementation to the one described in the paper it would be possible to rescan a mapped area if it is traversed again, and by that refine the mapping. This is requires however very good localisation techniques.

1.0.2 Deployment

The deployment strategy is an important part of this project as it defines how effective the robot will cover the target area which will define how long it will take to scan and map the whole area. One research paper which was read 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 [2]. The paper uses a solution which is based on small comm nodes deployed by the robot, to make the solution fitting for this project the developer would have to use multiple E-Pucks and define some of these as communication nodes which remain on a fast position and guide the "scout" E-puck based on area which have been least visited by the other robots.

However since multi-robot usage is planned as an addition if enough time is available this deployment strategy is not fitting for the major part of work.

1.1 Analysis

Taking into account the problem and what you learned from the background work, what was your analysis of the problem? How did your analysis help to decompose the problem into the main tasks that you would undertake? Were there alternative approaches? Why did you choose one approach compared to the alternatives?

There should be a clear statement of the objectives of the work, which you will evaluate at the end of the work.

In most cases, the agreed objectives or requirements will be the result of a compromise between what would ideally have been produced and what was felt to be possible in the time available. A discussion of the process of arriving at the final list is usually appropriate.

1.2 Process

You need to describe briefly the life cycle model or research method that you used. You do not need to write about all of the different process models that you are aware of. Focus on the process

model that you have used. It is possible that you needed to adapt an existing process model to suit your project; clearly identify what you used and how you adapted it for your needs.

Chapter 2

Design

This section will describe the design as of the date of the project outline description.
This is the design which will be followed and tried to implement, however this is more of a guideline rather than exact plan since at this moment it is uncertain what the API and simulator are able to do and what is possible to implement inside the given time.

2.1 Environment Design

The final environment for this project will consist of one large room with obstacles placed in it. The program will be tested on a number of different sized rooms with different amount and sized obstacles in it.
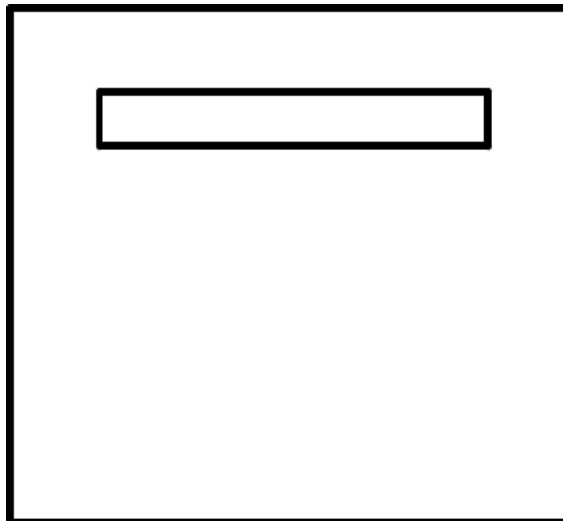

Figure 2.1: Environment Design

2.2   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.

2.3   Deployment

It is at this moment still undecided which deployment strategy will be implemented.
There are 2 deployment strategies which are based on the background research which has been done.

2.3.1   Considered Deployment algorithms

One which is based on a random walk though the environment which will turn to a random heading when an obstacle has been reached.
This approach could be combined with a wall following algorithm which would trigger a random walk when the same position is reached again. This would allow for the complete traverse of a obstacle/wall. This would require a good localisation solution as without it the robot will be stuck inside an eternal loop.

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.

2.3.2   Adopted design

It was decided to adopt and implement the more controlled movement pattern which can be seen in figure 2.2. dasdasdasd
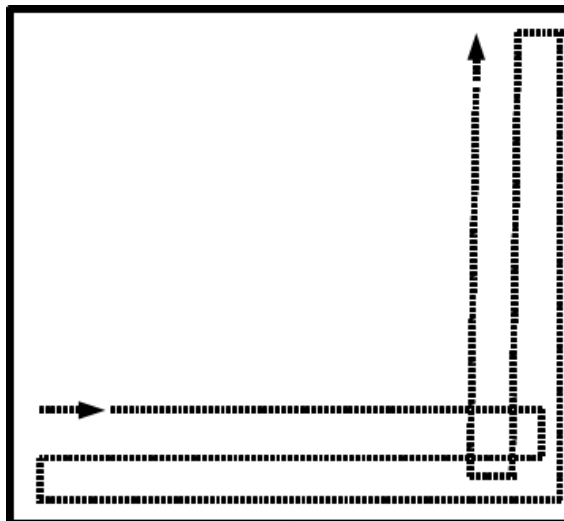
Figure 2.2: Controlled movement Pattern

Chapter 3

Implementation

3.1 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.

3.1.1 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.

3.1.2 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.

---

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

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 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.

3.2   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.

3.2.1   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.

3.3   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.

3.3.1   Moving Forward

The code for this function can be found in Appendix B, section 2.1.
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 the robot 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.

3.3.2   Turn a given angle

The code for this function can be found in Appendix B, section 2.2.

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.

```c
/**
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);
}
```

3.4 Localisation using Odometry

After the studying the optometry functions provided and implementing the movement algorithms, it was time to implement the localisation using odometry calculations.
The odometry functions I implemented are used for localisation the robot inside the environment

and finding it heading. The functions only require the starting point and localisation of the robot, and are then apple to calculate the movement and rotation of the robot with every movement done, within a certain degree of accuracy. The uncertainty in accuracy is based on the friction which get simulated inside the simulator.

These functions are large similar to the ones provided with the Webots™ interface, however some minor changes has been done.

Similar to the way I have described the movement algorithms I am going to describe the major functions as well as post the code for each function.

3.4.1   Initialising the Odometry algorithms

To initialize the odometry algorithms, 2 functions are used.

The first function, *odometry_track_start* takes a *odometrtyTrackStruct*, which is defined in the class which calls the function, as parameter. It will then acquire the encoder positions of the robot and call the next function, *odometry_track_start_pos* which set's the starting values, including the encoder positions inside the odometry struct.

```
struct odometryTrackStruct {
      struct {
            float wheel_distance;
            float wheel_conversion;
      } configuration;
      struct {
            int pos_left_prev;
            int pos_right_prev;
      } state;
      struct {
            float x;
            float y;
            float theta;
      } result;
};

int odometry_track_start(struct odometryTrackStruct * ot);
int odometry_track_start_pos(struct odometryTrackStruct * ot, double*
   dEncPos);
void odometry_track_step(struct odometryTrackStruct * ot);
void odometry_track_step_pos(struct odometryTrackStruct * ot, double*
   dEncPos);
```

Also the distance travel when a wheel turns and the wheel conversion are calculated, used for this are parameters acquired during calibrations done in the Webots™ example programs(the values are the same as it is the same virtual robot model) and the E-Puck documentation.

3.4.2 Updating the Odometry values

To update the odometry values of the struct the function *odometry_track_step* is called. When this function is called inside another function a number of things happen.

The current encoder positions for the stepper motors are fetched, and used as a parameter inside the *odometry_track_step_pos* function call.

Inside this function the new X, Y coordinates and the rotation of the robot are calculated. This is achieved by first calculating the difference between the current encoder positions and the encoder previous encoder positions saved inside the struct. This difference is then multiplied by the wheel conversion, which gets calculated inside the initialization step.

The result of this is used to calculated the wheel movement of the left and right wheel, results which are used to calculate the rotation of the robot. The next step is to calculate the new X and Y coordinates of the robot, based on the sum of done left and right wheel movement and math calculations using the robots rotation. At the end the calculated X and Y coordinates as well as the rotation value are used to update to struct. And the current encoder positions are saved inside the buffer for later calculations.

```c
void odometry_track_step(struct odometryTrackStruct * ot){
    double* point_dEncPos;

    point_dEncPos = get_encoder_positions();
    odometry_track_step_pos(ot, point_dEncPos);
}

void odometry_track_step_pos(struct odometryTrackStruct * ot, double*
   dEncPos){
    int delta_pos_left, delta_pos_right;
    float delta_left, delta_right, delta_theta, theta2;
    float delta_x, delta_y;

    //calculate the difference in position between the previous and
        current encoder positions.
    delta_pos_left = dEncPos[0] - ot->state.pos_left_prev;
    delta_pos_right = dEncPos[1] - ot->state.pos_right_prev;

    //calculate the rotation based on the displacement of the
        stepper motors
    delta_left = delta_pos_left * ot->configuration.wheel_conversion;
    delta_right = delta_pos_right *
        ot->configuration.wheel_conversion;
    delta_theta = (delta_right - delta_left) /
        ot->configuration.wheel_distance;

    // calculate the x and y displacement
    theta2 = ot->result.theta + delta_theta * 0.5;
    delta_x = (delta_left + delta_right) * 0.5 *cosf(theta2);
    delta_y = (delta_left + delta_right) * 0.5 * sinf(theta2);

    //update the x, y and theta of the struct
    ot->result.x += delta_x;
    ot->result.y += delta_y;
    ot->result.theta += delta_theta;
```

```
    if(ot->result.theta > M_PI){
            ot->result.theta -= 2*M_PI;
    }
    if(ot->result.theta < -M_PI){
            ot->result.theta += 2*M_PI;
    }

    //save current encoder positions to the global buffer
    ot->state.pos_left_prev = dEncPos[0];
    ot->state.pos_right_prev = dEncPos[1];
}
```

3.4.3 Implementation

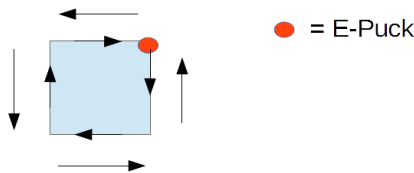After these functions were created I tested them using simple test simulations I wrote.

Figure 3.1: Movement algorithm test pattern

Figure 4 shows a simple movement pattern which I used to calibrate the minimum turn distances and tested that the *move_forward()* and *turn_angle()* methods worked as intended.
How it works is the E-Puck starts on 1 corner of a square(here I used the floor of the simulator as it has chessboard representation) and move forwards to the other end of the square, turn 90 degrees and so on until it reaches it start point. It then turns around and follows the same pattern back to its original starting position. This tests allows me to notice odometry error in the rotations and forward movement, after a while it would turn/move to far or not far enough and thereby not reach it accurate start point. Noticing these errors allowed me to calibrate my the speed and distance where the algorithms will slow done the movement to minimize this error as much as possible, see section 3.2 for more information on this.

The accuracy of this test can be increased by using larger movement areas, I used changing 2x2, 4x4 and 3x3 square areas of the chessboard floor to check that my error was at minimal as possible. The test is based on the "University of Michigan Benchmark" or UMBmark[2].

---

[2]urlhttp://www.cs.columbia.edu/ allen/F13/NOTES/borenstein.pdf

Chapter 4

Testing

Detailed descriptions of every test case are definitely not what is required here. What is important is to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.

Have you tested your system on real users? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

The following sections indicate some areas you might include. Other sections may be more appropriate to your project.

4.1   Overall Approach to Testing

4.2   Automated Testing

4.2.1   Unit Tests

4.2.2   User Interface Testing

4.2.3   Stress Testing

4.2.4   Other types of testing

4.3   Integration Testing

4.4   User Testing

Chapter 5

Future Plans

5.1 Communication

While it is possible to transfer information easily between robots since a simulator is used one keypoint was to try 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.

It has not been decided how this would be implemented since as of now it is not sure if the E-puck models implemented inside the simulator are able to send signals to other robots. While it is possible to transfer signals through the E-puck's laser sensors it is not know if this would actually be a better implementation than using radio transmitters and receivers.

The theory of what could be implemented 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.
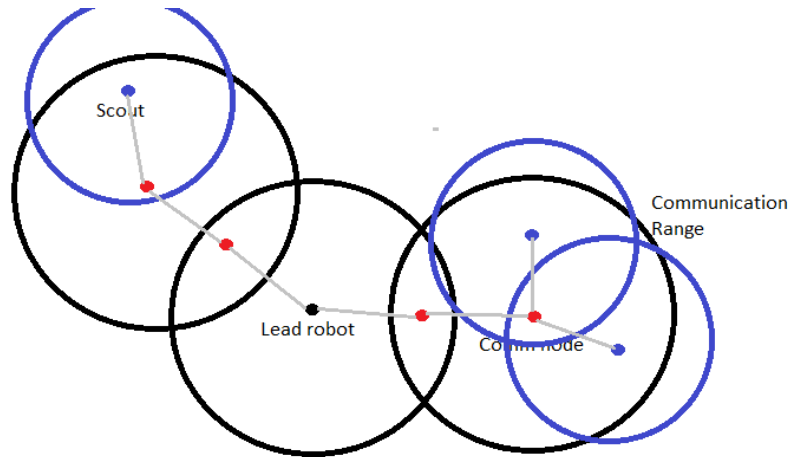

Figure 5.1: An example of the communication link

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.

Chapter 6

Evaluation

Examiners expect to find in your dissertation a section addressing such questions as:

- Were the requirements correctly identified?

- Were the design decisions correct?

- Could a more suitable set of tools have been chosen?

- How well did the software meet the needs of those who were expecting to use it?

- How well were any other project aims achieved?

- If you were starting again, what would you do differently?

Such material is regarded as an important part of the dissertation; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things and room for improvement with any project. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

Review the discussion on the Evaluation section from the lectures. A recording is available on Blackboard.

Appendices

Appendix A

Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. The key requirement is that we understand what is your original work and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

As an example, you might include a definition such as:

Apache POI library The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the clients existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [**?**]. The library is released using the Apache License [**?**]. This library was used without modification.

Appendix B

Code samples

2.1 Moving forward a given distance

This function moved the robot a given distance with a given speed.
The robot will slow down to a minimum speed on a minimum difference to the wanted position.
This method will update the global odometry information.

```c
#define WHEELBASE 0.058
#define INCREMENT_STEP 1000 //how many steps the motor takes for a
    full wheel rotation
#define MIN_DIST 20.0f //minimum difference to the new heading
#define MIN_SPEED 10.0f //speed when slowing down

#ifndef M_PI
    #define M_PI 3.14159265358979323846264433832795L
#endif

/**
Function to move the robot forward a given distance at a given speed
*/
void move_forward(double dSpeed, double dDist, struct
    odometryTrackStruct * ot){
    double dStepCount = 0.0f;
    double dStopPosLeft = 0.0f;
    double dStopPosRight = 0.0f;
    double *point_dEncPos;

    if((dDist > 0.0f) && (dSpeed > 0.0f)){
        //calculate the number of steps
        dStepCount = (INCREMENT_STEP/(M_PI * WHEEL_DIAMETER / 2))
            * dDist;

        /*read the current encoder positions of both wheels and
        calculate the encoder positions when the robot has to
        stop at the given distance ... */
        point_dEncPos = get_encoder_positions();

        dStopPosLeft = point_dEncPos[0] + dStepCount;
```

```
                        dStopPosRight = point_dEncPos[1] + dStepCount;

                        //compute odometry data
                        point_dOdometryData = compute_odometry_data();
                        odometry_track_step(ot);


                        //set speed
                        set_motor_speed(dSpeed, dSpeed);

                        //step tolereance test
                        while((point_dEncPos[0] < dStopPosLeft) &&
                            (point_dEncPos[1] < dStopPosRight)){
                                //get odometry data
                                point_dOdometryData = compute_odometry_data();
                                odometry_track_step(ot);
                                //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);
                                }
                        }
                }
                stop_robot();

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

2.2 Turning a given Angle

This function turns the robot a given amount of degrees with a given amount of speed.
The robot will slow down to a minimum speed on a minimum threshold to the wanted heading.

```
#define LEFT_DIAMETER 0.0416
#define RIGHT_DIAMETER 0.0404
#define WHEEL_DIAMETER (LEFT_DIAMETER + RIGHT_DIAMETER)
#define WHEELBASE 0.058
#define INCREMENT_STEP 1000 //how many steps the motor takes for a
    full wheel rotation
#define MIN_DIST 20.0f //minimum difference to the new heading
#define MIN_SPEED 10.0f //speed when slowing down

/**
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_DIAMETER / 2);

            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);
}
```

Annotated Bibliography

[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] 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

[3] 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

[4] 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

[5] 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

[6] 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

[7] 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

[8] 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

[9] 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

[10] 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

[11] S. Thrun, "Learning Occupancy Grids with Forward Models," in *In Proceedings of the Conference on Intelligent Robots and Systems (IROSâ2001*, 2001, pp. 1676–1681. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.6014

[12] 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

[13] 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

[14] 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