Module 35 Artificial Intelligence & Robotics

Module title	Artificial Intelligence & Robotics
Module NFQ level (only if an NFQ level can be demonstrated)	8
Module number/reference	BSCH-AIR
Parent programme(s)	Bachelor of Science (Honours) in Computing Science
Stage of parent programme	Award stage
Semester (semester1/semester2 if applicable)	Semester 2
Module credit units (FET/HET/ECTS)	ECTS
Module credit number of units	5
List the teaching and learning modes	Direct, Blended
Entry requirements (statement of knowledge, skill and competence)	Learners must have achieved programme entry requirements.
Pre-requisite module titles	BSCH-CP, BSCH-OOP, BSCH-CD
Co-requisite module titles	None
Is this a capstone module? (Yes or No)	No
Specification of the qualifications (academic, pedagogical and professional/occupational) and experience required of staff (staff includes workplace personnel who are responsible for learners such as apprentices, trainees and learners in clinical placements)	Qualified to as least a Bachelor of Science (Honours) level in Computer Science or equivalent and with a Certificate in Training and Education (30 ECTS at level 9 on the NFQ) or equivalent.
Maximum number of learners per centre (or instance of the module)	60
Duration of the module	One Academic Semester, 12 weeks teaching
Average (over the duration of the module) of the contact hours per week	3
Module-specific physical resources and support required per centre (or instance of the module)	One class room with capacity for 60 learners along with one computer lab with capacity for 25 learners for each group of 25 learners

Analysis of required learning effort		
	Minimum ratio teacher / learner	**Hours**
Effort while in contact with staff		
Classroom and demonstrations	1:60	18
Monitoring and small-group teaching	1:25	18
Other (specify)		
Independent Learning		
Directed e-learning		
Independent Learning		57
Other hours (worksheets and assignments)		32
Work-based learning – learning effort		
Total Effort		125

Allocation of marks (within the module)					
	Continuous assessment	**Supervised project**	**Proctored practical examination**	**Proctored written examination**	**Total**
Percentage contribution	40%	60%			100%

Module aims and objectives

The aim of this module is to enable the learner to embody an artificial intelligent agent in the physical or virtual world. The virtual hardware or physical hardware, programmed behaviours, and algorithms will be toughly understood, implemented, and customized by the learners.

This aim will be met through the pursuit of the following objectives:

- To familiarize learners with a number of AI problems including probabilistic inference, planning and search, localization, tracking and control.
- To equip learners with skills in representing these problems and their solutions with appropriate notation.
- To assist learners in expanding their programming competencies to include a programming language suitable for implementing AI behaviours and porting them to physical or virtual robotics hardware.
- To support learners in the sequence of: identifying the problem type, selecting the appropriate AI algorithmic solution and implementing this AI algorithmic solution in an appropriate programming language.
- To expose learners to a range of physical and virtual robotic hardware and assist them in evaluating it suitability to perform AI tasks.
- To provide learners with a framework to evaluate the performance of robots performing AI tasks.

Minimum intended module learning outcomes
On successful completion of this module, the learner will be able to:

1. Discuss the major challenges facing AI.
2. Select and apply appropriate AI algorithms to solve real world problems.
3. Represent logical and geometric problems with appropriate notation.
4. Evaluate the suitability of physical and simulated robotic hardware to perform AI tasks.
5. Programme physical or simulated robots to perform AI tasks.
6. Deploy and evaluate robotic performance of AI tasks

Rationale for inclusion of the module in the programme and its contribution to the overall MIPLOs
The growth of the technology sector in the area of AI and robotics can be exemplified through many of today's devices including the control system in self-drive cars and drones. Investment in AI based technology products is increasing resulting in an increased demand for computer scientists in this area.

In this module learners take a number of modules in which they construct programs to solve well defined problems many of which have optimal solutions. This module aims to expand the set of problems to include those which may not have optimal solutions or have solutions which depend on the perception of external factors or circumstances.

The module empowers learners to see their programming solutions manifest in the physical or virtual world.

Appendix 1 of the programme document maps MIPLOs to the modules through which they are delivered.

Information provided to learners about the module
Learners receive a programme handbook to include module descriptor, module learning outcomes (MIMLO), class plan, assignment briefs, assessment strategy, and reading materials.

Module content, organisation and structure
AI Introduction

- What is AI
- The Foundations and History of AI

- State of the Art
- What is Planning
- Search Localization Tracking
- Control

Robotics Introduction

- Welcome
- Quadrotors
- Flying Principle
- Quadrotor Research

Linear Algebra & Geometry

- Linear Algebra
- 2D Geometry
- 3D Geometry

Sensors

- Sensors

Actuators & Control

- Motors & Controllers
- Feedback Control
- Kinematics & Dynamics
- PID Control

Probability Theory, Bayes Rule & Grid Filter

- State Estimation
- Probability Theory
- Bayes Rule

Probabilistic State Estimation

- Bayes Filter
- Histogram Filter
- Kalman Filter
- Extended Kalman Filter
- Particle Filter

Search

- A*
- Dynamic Programming

Visual Motion Estimation

- 2D Motion Estimation
- Visual Odometry
- Resume Course

Visual SLAM

- Visual Navigation with a Parrot ARdrone
- Tracking and Mapping with Signed Distance Functions
- Direct Methods for Visual SLAM

Module teaching and learning (including formative assessment) strategy
The module is taught as a combination of lectures and lab sessions. The lecture sessions assist the learner in exploring the theoretical underpinnings of AI & Robotics. The practical lab sessions give learners the opportunity to implement AI algorithms and embed them in robots. Learners also experience first-hand how their robots interact with the world based on their programmed AI behaviours. Through prescribed experimentation, and learner collaboration, learners embody their AI knowledge in robots.

Assessment is divided into 3 elements. There are two take home assignments that assess the learner's competency in specific areas of the syllabus, while giving them increasing larger problems to solve. There is a group project to be completed which tests the learners' understanding of the theoretical material and allows them to realize a robot exhibiting AI behaviours.

Timetabling, learner effort and credit
The module is timetabled as one 1.5-hour lectures and one 1.5-hour lab per week.

The number of 5 ECTS credits assigned to this module is our assessment of the amount of learner effort required. Continuous assessment spreads the learner effort to focus on small steps before integrating all steps into a realizing a robot exhibiting AI behaviours.

There are 36 contact hours made up of 12 lectures delivered over 12 weeks with classes taking place in a classroom. There are also 12 lab sessions delivered over 12 weeks taking place in a fully equipped computer lab. The learner will need 57 hours of independent effort to further develop the skills and knowledge gained through the contact hours. An additional 32 hours are set aside for learners to work on worksheets and assignments that must be completed for the module.

The team believes that 125 hours of learner effort are required by learners to achieve the MIMLOs and justify the award of 5 ECTS credits at this stage of the programme.

Work-based learning and practice-placement
There is no work based learning or practice placement involved in the module.

E-learning
The college VLE is used to disseminate notes, advice, and online resources to support the learners. The learners are also given access to Lynda.com as a resource for reference.

Module physical resource requirements
Requirements are for a classroom for 60 learners equipped with a projector, and a 25 seater computer lab for practical sessions with access to Python and Java development environments and associated libraries (this may change should more suitable technologies become available).

Reading lists and other information resources
Recommended Text
Russell, S. (2016) *Artificial intelligence: a modern approach, global edition.* Upper Saddle River: Pearson Education Limited.

Martinez, A., Fernández, E. (2015) *Learning ROS for robotics programming.* Birmingham: Packt Publishing

Secondary Reading:
Joseph, L. (2018) *Learning Robotics Using Python*. Birmingham: Packt Publishing

Parot.com (2016) Developer Documentation
http://developer.parrot.com/docs/SDK3/ [3]

Montemerlo, M. and Thrun, S. (2007) *FastSLAM*. Berlin: Springer.

Specifications for module staffing requirements
For each instance of the module, one lecturer qualified to at least Bachelor of Science (Honours) in Computer Science or equivalent, and with a Certificate in Training and Education (30 ECTS at level 9 on the NFQ) or equivalent.. Industry experience would be a benefit but is not a requirement.

[3] Last accessed 26/07/2018

Learners also benefit from the support of the programme director, programme administrator, learner representative and the Student Union and Counselling Service.

Module Assessment Strategy

The assignments constitute the overall grade achieved, and are based on each individual learner's work. The continuous assessments provide for ongoing feedback to the learner and relates to the module curriculum.

No.	Description	MIMLOs	Weighting
1	Two take home assignments that assess the learner's competency in specific areas of the syllabus	1,2,3	40%
3	Project to be completed with a partner.	1,2,4,5,6	60%

All repeat work is capped at 40%.

Sample assessment materials

Note: All assignment briefs are subject to change in order to maintain current content.

Course	BSCH
Stage / Year	4
Module	AI & Robotics
Semester	2
Assignment	Assignment 1
Date of Title Issue	x/x/x
Assignment Deadline	x/x/x (2 weeks after issue)
Assignment Submission	Upload to Moodle
Assignment Weighting	20% of module

Submission Details:

Please be sure to submit

- A single zipped file containing
    - This document complete with answers, mathematical calculations can be inserted using image files of a small size.
    - Code files for appropriate questions.

Section 1. Representations of Rotations – 25%

With the rotation visualizer provided on the course homepage you can animate your rotations:

- animateRotationMatrix(0,-1,0,1,0,0,0,0,1)
- animateEuler(0, 90, 0)
- animateAxisAngle(1,0,0,90)
- animateQuaternion(0,0,1,0) [note that the parameters here are animateQuaternion(qx,qy,qz,qw) as opposed to the exercise below!]

Rotation Matrices

1. Your quadrotor has crashed and is lying tipped over facing towards you in the field. What rotation do you have to perform to align it with your view again, i.e., with the camera facing forward and thus away from you? Enter the rotation matrix (format: "1,0,0; 0,1,0; 0,0,1"). Show your solution.

(3 mark)

2. How many more parameters do 3D rotation matrices have compared to minimal representations. Explain your answer.

(3 mark)

Euler Angles

3. How many different Euler angle conventions are there? Explain your answer.

(3 mark)

4. How would the rotation above be parametrized with Euler angles in the roll pitch yaw convention? Enter the RPY angles **in degree** (format: "30,40,50 "). Explain your answer.

(3 mark)

Euler to Rotation Matrix

5. Given the RPY Euler angles **in degree** (90, 0, 90) compute the rotation matrix (format: "1,0,0; 0,1,0; 0,0,1"). Show your calculations.

(3 mark)

Angle/Axis Representation

6. How many parameters has the Angle-Axis representation. Justify your answer.

(3 mark)

Advantages of Representations

7. Which representation are easy to concatenate?
   a. Rotation Matrices
   b. Euler Angles
   c. Angle-Axis
   d. Quernions

(2 mark)

8. Which representations can be minimal?
   e. Rotation Matrices
   f. Euler Angles
   g. Angle-Axis
   h. Quernions

(2 mark)

9. Which representations are easy to invert?
   i. Rotation Matrices
   j. Euler Angles
   k. Angle-Axis
   l. Quernions

(2 mark)

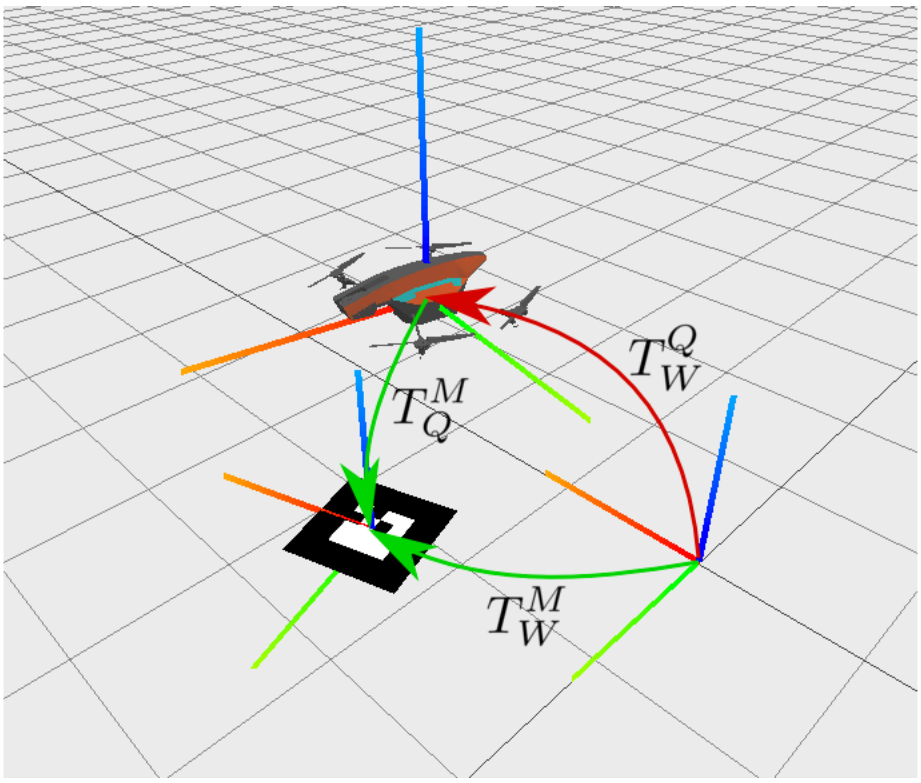
10. Which representation is unique? (1mark)
    m. Rotation Matrices
    n. Euler Angles
    o. Angle-Axis
    p. Quernions

(1 mark)

Section 2. 3D Transformations – 25%

1. In this exercise we want to compute the position of the quadrotor from observations of visual markers. The markers are detected in images recorded from the downfacing camera of the quadrotor. We also assume the position of the markers in the world are known.

   The following image illustrates the relations between the world, quadrotor and marker coordinate frames. The transformations from the world to the marker T_W^M and from the quadrotor to the marker T_Q^M are known. Your task is to compute the transformation from the world to the quadrotor T_W^Q given the two other transformations.



In the code below implement the predefined `compute_drone_pose` function. Its parameters are the global marker pose $T_W{}^M$ and the observed marker pose $T_Q{}^M$. Both are instances of the `Pose3D` class. It has to return the quadrotor pose $T_W{}^Q$.

```python
import numpy as np

class Pose3D:
    def __init__(self, rotation, translation):
        self.rotation = rotation
        self.translation = translation

    def inv(self):
        '''
        Inversion of this Pose3D object

        :return inverse of self
        '''
        # TODO: implement inversion
        inv_rotation = self.rotation
        inv_translation = self.translation

        return Pose3D(inv_rotation, inv_translation)

    def __mul__(self, other):
        '''
        Multiplication of two Pose3D objects, e.g.:
            a = Pose3D(...) # = self
            b = Pose3D(...) # = other
            c = a * b       # = return value

        :param other: Pose3D right hand side
        :return product of self and other
        '''
        # TODO: implement multiplication
        return Pose3D(self.rotation, self.translation)

    def __str__(self):
        return "rotation:\n" + str(self.rotation) + "\ntranslation:\n" + str
(self.translation.transpose())

def compute_quadrotor_pose(global_marker_pose, observed_marker_pose):
    '''''
    :param global_marker_pose: Pose3D
    :param observed_marker_pose: Pose3D

    :return global quadrotor pose computed from global_marker_pose and obser
ved_marker_pose
    '''
    # TODO: implement global quadrotor pose computation
    global_quadrotor_pose = None

    return global_quadrotor_pose
```

(25 marks)

Section 3. Probability Theory – 25%

Enter all decimal numbers with at least **two** digits after the decimal points. More digits are always allowed. Examples:

- 0.314 -> 0.31
- 5.675 -> 5.68
- 0.56 -> 0.56
- 0.2 -> 0.2

Probability

1. You have built two quadrotors guarding your house. From earlier experiments with a single quadrotor, we observed that the probability mass function describing the presence of a quadrotor is $P(X)=(0.2,0.2,0.6)$ where $X\in\{house,street,backyard\}$. For this exercise, we assume the locations of both quadrotors are independent of each other.What is the probability both quadrotors are in the backyard at the same time?

(6 mark)

Conditional Probability

2. Assume you have installed a sophisticated vision system on your quadrotor to recognize doors, i.e., $P(\text{door is detected}|\text{quadrotor is in front of door})=0.9$. However, navigating through a door only suceeds with a probability of 0.7, i.e., $P(\text{success}|\text{door is detected})=0.7$.What is the probability of successfully detecting and traversing a door when the quadrotor is in front of a door?

(6 mark)

Joint Probability

3. Now let's further assume, that there is a single charging station in the backyard. To prevent bumping into each other, the quadrotors are

communicating with each other to ensure that they are not flying in the same area. As a result, their locations are not independent of each other anymore. The resulting joint probability table now looks as follows:

P(X,Y)	X=House	X=Street	X=Backyard
Y=House	0.0	0.1	0.2
Y=Street	0.1	0.0	0.2
Y=Backyard	0.2	0.2	0.0

What is the probability of (at least) one quadrotor being in the backyard?

(6 mark)

Bayes Rule

4. Imagine a quadrotor seeks its charging station which is marked with many bright lamps.

   The quadrotor is equipped with a binary brightness sensor that can measure either bright or $\neg\mathrm{bright}$.

   The binary world state is either home or $\neg\mathrm{home}$ with the prior $\mathrm{P(home)}=0.2$.

   Moreover we know:

   $\mathrm{P(bright|home)}=0.8$

   and

   $\mathrm{P(bright|\neg home)}=0.3$

   Assume the robot observes light, what is the probability $\mathrm{P(home|bright)}$ that it is above the charging base?

(7
ma
rks)

Section 4. Statistics – 25 %

Mean, Variance & Standard Deviation

1. Please compute the sample mean, sample variance and standard deviation of the following time series: $-1,2,3,1,1$

(7 marks)

Statistics with Numpy

2. In this exercise you are given the time series of a (stationary) quadrotor equipped with a GPS receiver.

   Please calculate the statistics with numpy.

   Note: you can use the numpy functions `mean`, `var`, `cov`. Use the two parameter version of `cov`, i.e., to compute the covariance of two vectors a and b use `numpy.cov(a, b)`.

```python
import numpy as np

def compute_means(lat,lon,alt):
    # TODO: implement mean computation
    mean_lat = None
    mean_lon = None
    mean_alt = None
    return (mean_lat,mean_lon,mean_alt)

def compute_vars(lat,lon,alt):
    # TODO: implement variance computation
    var_lat = None
    var_lon = None
    var_alt = None
    return (var_lat,var_lon,var_alt)

def compute_cov(lat,lon,alt):
    # TODO: implement covariance computation

    cov_lat_lon = None
    cov_lon_alt = None
    cov_lat_alt = None
    return (cov_lat_lon,cov_lon_alt,cov_lat_alt)
```

(10 marks)

Accuracy of GPS

3. The manufacturer of a high quality GPS receiver claims that his device achieves the following accuracies:
   - $\mathrm{stddev(latitude)}=5.4477688039°\cdot 10{-}06$
   - $\mathrm{stddev(longitude)}=9.89285349017°\cdot 10{-}06$

   For the latitude 48.262° of Munich,

   - 1° of latitude corresponds to 111195.38097356868 m, and
   - 1° of longitude corresponds to 74246.69042433369 m.

   What is the corresponding standard deviation of the GPS receiver in latitude in m in Munich?

   What is the corresponding standard deviation of the GPS receiver in longitude in m in Munich?

   (8 marks)

Course	BSCH
Stage / Year	4
Module	AI & Robotics
Semester	2
Assignment	Assignment 2
Date of Title Issue	x/x/x
Assignment Deadline	x/x/x (2 weeks after issue)
Assignment Submission	Upload to Moodle
Assignment Weighting	20% of module

Question 1: Biasdness of Particle Filters – 25%

In class, we discussed in length the fact that Monte Carlo Localization (and particle filters) are biased for finite sample sets, as a result of the way particles are resampled. In this question, you are asked to quantify this bias.
To simplify things, consider a world with 4 possible robot locations: $S = \{s_1, s_2, s_3, s_4\}$

s_1	s_2
s_3	s_4

Initially, we draw N >= 1 samples uniformly from among those locations—-as usual, it is perfectly acceptable if more than one sample is generated for any of the locations *S*. Let now *z* be our first actual sensor measurement. Suppose that is characterized by the following conditional probabilities:

$$\begin{aligned} p(z|s_1) &= 0.8 & \quad p(\neg z|s_1) &= 0.2 \\ p(z|s_2) &= 0.4 & \quad p(\neg z|s_2) &= 0.6 \\ p(z|s_3) &= 0.1 & \quad p(\neg z|s_3) &= 0.9 \\ p(z|s_4) &= 0.1 & \quad p(\neg z|s_4) &= 0.9 \end{aligned}$$

As explained in class, these probabilities are used to generate importance factors, which are subsequently normalized and used for resampling. For simplicity, let us assume we only generate one new sample in the resampling process, regardless of *N*. This sample might correspond to any of the four locations in *S*. Thus, the sampling process defines a probability distribution over *S*.
Questions:

1.1 What is the resulting probability distribution over for this new sample? Answer this question separately for *N=1, …., 10,* and for $N = \infty$. Your answers have to be exact (truncation errors are acceptable).

1.2 What is the KL divergence between those probability distributions and the "true" posterior, derived from Bayes filters? Again, answer this question separately for *N=1, …., 10,* and for $N = \infty$. The KL divergence between a $\widehat{p}$ distribution and a "true" distribution p is given by

$$KL(\hat{p}, p) \quad = \quad \sum_i \hat{p}_i \log \frac{\hat{p}_i}{p_i}$$

Again, your answers have to be exact (up to truncation errors).

1.3 Prove the correctness of your answers for *N=1, N= 2*, and *N* = ∞.

1.4 What modification of the problem formulation would guarantee that the specific estimator above is unbiased even for finite values of *N*? Provide at least two such modifications (each of which should be sufficient).

Hint: I wrote a deterministic *program to calculate some of these results.*

Question 2: Kalman Filter Localization – 25%

Implement a Kalman-style filter for the following problem. You have a robot whose state is x, y, θ. Initially, it is at pose 0; 0; 0 with no uncertainty. It attempts to move forward at 1 m/s while rotating at 0.1 radian/s for 10 seconds. This command is executed in an open-loop fashion, that is, it does not change its control based on sensor feedback. It has some banana-shaped noise in its motion update (pick any reasonable distribution).
Every second it receives a GPS sensor observation; that is, it gets told its x; y position (but not its orientation θ) corrupted by normally-distributed independent noise of standard deviation 2 meters. You may linearize the problem according to any of the methods we discussed in class, such as the extended Kalman filter or the unscented Kalman filter.
Turn in (1) a mathematical description of your algorithm along with its derivation (no need to derive Kalman filters, but you might want to state them so we understand which version you used), (2) a code listing, and (3) an actual graph of the robot's mean position and uncertainty vs. time during some sample runs obtained with your implemented code. This graph should show uncertainty ellipses for the robot in 1-second intervals.
At the end of the 10 seconds, has the robot found out any information about its orientation at time t = 3s?
Why or why not?

Question 3: Beyond Probabilities? - 25%

The key idea of probabilistic robotics is to maintain probability distributions over unknown quantities such as robot poses and maps. Can you imagine situations where a probability distribution might be insufficient to accurately characterize the state of knowledge? If yes, describe one. If not, argue why no such situation might exist.

Question 4: EM for Mapping Forests – 25%

In class, we talked about how to use the expectation maximization (EM) algorithm for generating 3D maps from range measurements taken at known poses. In this question, you are asked to derive a similar algorithm, but with two differences:

1. All sensor measurements are in a single plane, i.e., we are back to a two-dimensional mapping problem. Since the robot poses are assumed to be known, it may be convenient to think of as a location in - space.
2. All objects in the world are trees with known radius. Since this is a two-dimensional problem, each tree will show up as a circle of radius in the final map.
3. For simplicity, let us assume that the number of trees is known a priori, and that the measurement noise is Gaussian. In particular, there is no need to consider other sources of noise or objects other than trees.

Your questions:

1. Provide (and derive) the generative model.
2. What is the expected log likelihood that is being maximized in EM? Please provide a derivation.
3. Derive all necessary equations for the E-step.
4. Lay out your solution for the M-step. If you can't find a closed form solution, you might provide an algorithm for improving the map (this is known as Generalized EM).

Suggestion: You might want to use or Tom Mitchell's book *Machine Learning* (McGraw Hill 1997. Simply follow the outline of the math provided there and modify it to accommodate circular objects with radius.

Course	BSCH
Stage / Year	4
Module	AI & Robotics
Semester	2
Assignment	Project with partner
Date of Title Issue	Start of Week 6
Assignment Deadline	End of Week 12
Assignment Submission	Upload to Moodle
Assignment Weighting	60% of module

Submission Requirements:

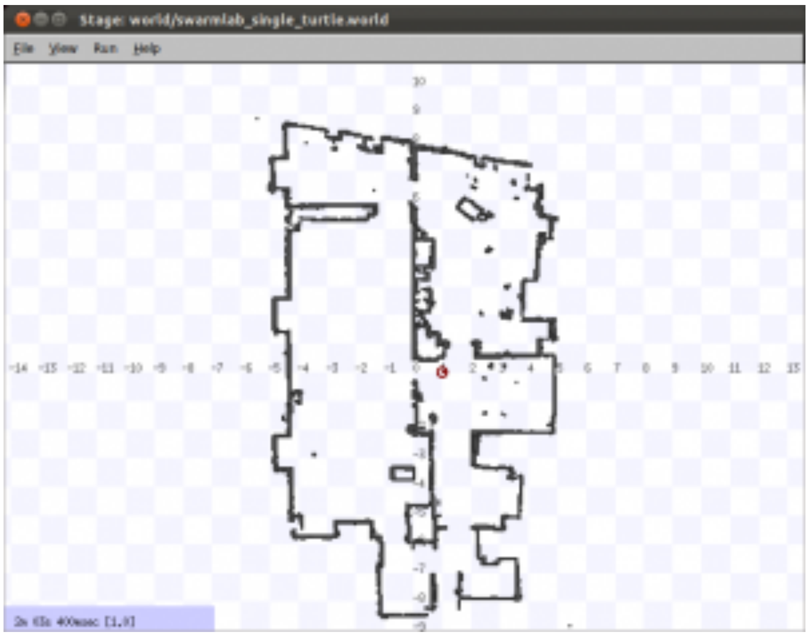
- You are **both** required to submit a single **identical** archive (zip) file to Moodle that has a filename of **<lastname1>_<firstname1>_<studentnumber1>_<lastname2>_<firstname2>_<studentnumber2>_project.zip** If the files are not identical (MD5 hash etc.) then you will be called to interview.
- It should contain the following
  - Source code
  - Documentation
  - Screen shots / video of the working implementation

Section 1 - Robot Operating System – 10 %

The Objective of this section is for you to become familiar with the Robot Operating System(ROS) software, which you will be using in future parts to program various algorithms for robots.
Before you can start this section you have to complete the following steps

1. Make sure you installed ROS and configured your workspace according to the installation instructions.
2. Follow all the beginner level tutorials from step 2 onward, to get yourself familiar with the ROS system.

In this section we will work with a simulated robot equipped with a laser range finder which scans it environment in a horizontal plane for obstacles. In addition to the standard ROS core packages you will use the 2D robot simulator called Stage. Stage simulates a population of mobile robots, sensors and objects in a two-dimensional bitmapped environment. Stage provides fairly simple, computationally cheap models of lots of devices rather than attempting to emulate any device with great fidelity.



We are going to create a random walk algorithm for a robot to drive through the lab without driving into obstacles.

The end effect of this section should be that the robot moves forward until it reaches an obstacle, than it rotates in place for a random amount of time and then moves forward again, etc.

We start with creating a simple ROS node called *random_walk*.

```
cd ~/catkin_ws/src

catkin_create_pkg random_walk roscpp geometry_msgs sensor_msgs

cd random_walk
```

Create a directory called */world* and unpack these world files in there. The files are the actual map which is used in stage, visualization of the robot and the physical robot and world description.
In a new terminal start the ROS core

```
roscore
```

Start Stage in the previous terminal (make sure you are located in */random_walk* folder)

```
rosrun stage_ros stageros world/lab_single_turtle.world
```

In the view menu of Stage you can enable the 'Data' option to see the sensor data of the robot. You can move your robot by clicking in stage and dragging it to another position.

Now, to be able to use some of the robot functionality go ahead and download teleoperation package for a simulated Turtlebot. Note that compilation of the teleoperation package requires *joy* package to be installed, this can be done through a standard package manager. In case you have any problems with the downloaded package, check the official "Turtlebot" project on Git and download the package from there. Unzip it in the *~/catkin/src* folder. Finally build the source code by typing:

```
cd ~/catkin_ws

catkin_make
```

With the package in place start teleoperation in a new terminal.

```
rosrun turtlebot_teleop turtlebot_teleop_key ~cmd_vel:=/cmd_vel
```

In case you receive a "not executable" message from the *rosrun*, make sure to add executable mode to the suggested file.
If you select the teleoperation terminal, you can use your keyboard to give your robot commands to drive it around.

To control the robot with our newly created node we prepared some example code. Download the example-code file and copy it to the /src directory of your *random_walk* package.
You need to add your source code to the make list of the compiler. To do so open your *CMakeLists.txt* file and make sure that under "Declare a c++ executable" you have a link to your source code:

```
add_executable(random_walk src/random_walk.cpp)
```

and that you link against catkin libraries:

```
target_link_libraries(random_walk

  ${catkin_LIBRARIES}

)
```

Build the code as mentioned above with *catkin_make* and run the code in another terminal

```
rosrun random_walk random_walk
```

You should start seeing range values being printed in the terminal, along with some timestamps. If you now move the robot with the keyboard or mouse, you will notice that the range value's change according to the presence of obstacles.

Your task is now to take the example code and adapt it in the given places to implement a simple random walk algorithm as described above. You are free to

extend the random walk algorithm to a more sophisticated one to achieve higher marks.
Automated Startup method

You can also automate the startup of the needed nodes by using a launch file, when using a launch file you don't need to start the roscore, this will be automatically done, furthermore the nodes that you specified will be started. To make a launch file, you create a file named *random_walk.launch* in your *random_walk* package and paste the next code in it.

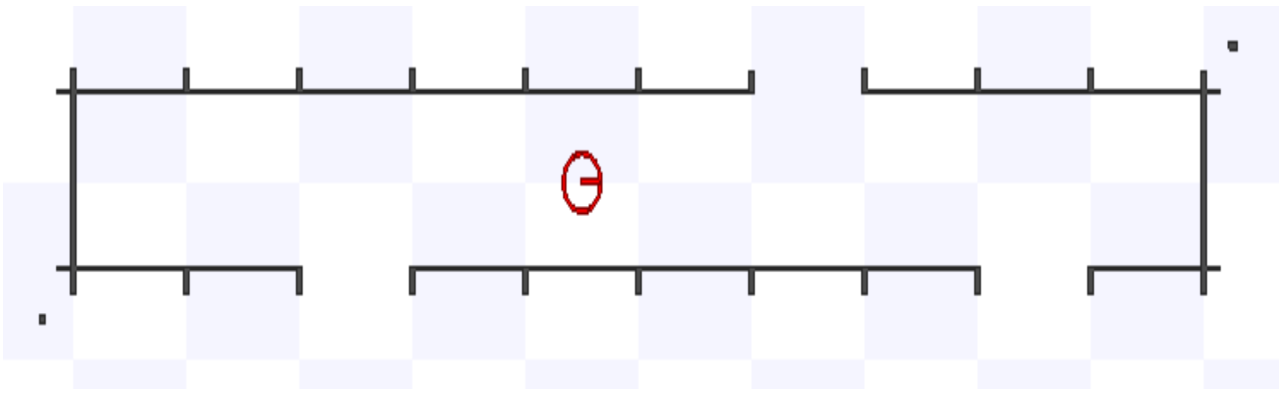
```
<launch>

  <node name="stageros" pkg="stage_ros" type="stageros" args="$(find random_walk)/world/lab_single_turtle.world" respawn="false" output="screen" />

  <node name="random_walk" pkg="random_walk" type="random_walk" output="screen" />

</launch>
```

Start your program now with

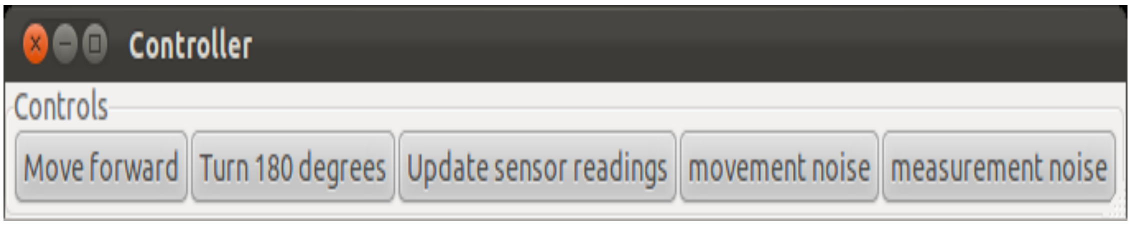
```
roslaunch random_walk random_walk.launch
```

Section 2 – Driving a Robot - 20%

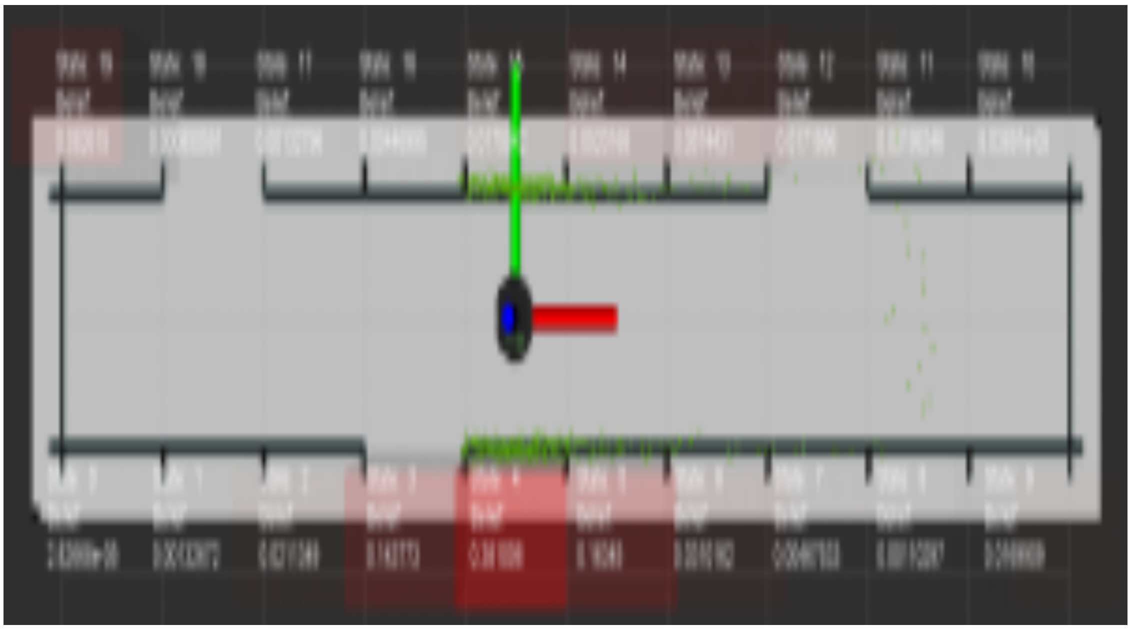
In this section the robot drives through a corridor with several doors in it and has to localize itself. To achieve this, you are going to implement a Bayes filter.



The robot can be controlled by an interface, called "controller" which has several buttons to perform some required actions.



The robot world is discretized in grid cells of 1m *x* 1m. There are 10 of these grid cells in this world. The robot moves forward by 1m by pressing the "Move" button, and turns 180 degrees by pressing the "Turn" button. This means the robot can move in both directions through the corridor and every grid-cell is represented by two belief-states (one facing left, one facing right). State 0 is at the upper right corner, then increasing to state 9 at the upper left, state 10 is at the lower left and increasing to state 19 in the lower right. So states 0..9 represent the beliefs for the robot facing left, and 10..19 represent the beliefs for the robot facing to the right. The measurements of the robots are also discretized to detect walls at a maximum distance of 1m (done by the *laser_to_wall* node). This means that at every position (state) the robot gets three possible measurements (e.g. *wall_right*, *wall_left*, *wall_front*).



To visualize the robot position and the belief of the robot for every possible state we visualize everything in RViz. You see the robot and see an overlay of the states onto the map. Next to the state number the belief probability is printed and the higher the probability, the less opaque the red marker gets.

Localization without uncertainty wouldn't be much fun, so you can enable/disable both movement and measurement noise with the controller buttons, by default noise is disabled. (When you pressed the button, you can check the noise state in the terminal.) The following measurement and movement models are used:

In the example code the general framework for running this simulation is provided, you have to fill in the gaps!

- Implement the measurement model in *updateSensing()*
- Implement the motion model in *updateMove()* and *updateTurn()*

Hints:

- Create an internal representation of the world to compare your measurements against.
- The three different sensor measurements can be treated with separate probabilities.
- View the tutorials on Bayes filters.

Practical tip:

– For those of you who would like to solve move/turn overshoot problem (particularly relevant if you run ROS on a VM), you might have to modify "Move" and "Turn" actions of the simulated robot so that it takes into account its own odometry. You might have to look into quaternion coordinate representation and see how you can grab current robot position/orientation at the time of movement.

Installation:

Download and unzip these packages into your *~/catkin/src* workspace directory. Go inside a terminal to the */nodes* directory of the *controller* and *laser_to_wall* packages and make the python files in these packages executable, otherwise you can't run them.

```
chmod +x filename.py
```

Create a new package called "bayes_fiter" with dependency's of "roscpp", "laser_to_wall", "controller", and "geometry_msgs". (see Section 1, how to create a package)

Copy the example-code file (bayes_filter.cpp) into the */src* directory of this package.

Add this source-file to the CMakeLists.txt (see Section 1, how to do this). Build your catkin workspace.

Inside the "bayes_world" directory we have created a launch file "bayes_world.launch", it starts the environment you need for this (see Section 1 how to run the launch file). It starts:

1. joint_state_publisher, needed for visualization purposes. It publishes messages about the robot's joint states.
2. robot_state_publisher, needed for visualization purposes. It publishes messages about the robot's body state.
3. Stage, with the required robot model and map.
4. *fake_localization*, needed for simulation purposes. Note: this package might have to be installed through the package manager.
5. *map_server*, that publishes the map so it can be used in RViz. Note: this package might have to be installed through the package manager.
6. *laser_to_wall*, which is a custom node that translates from laser data to *wall_front*, *wall_left*, *wall_right detection*, with a maximum distance of 1 meter.
7. *controller*, this custom nodes provides control buttons for moving the robot around, turning, measurements and enabling/disabling noise. Note: this

package might require installation of the wxPython through the package manager.

8. RViz, in which we visualize the robot position and more important, the belief states of the robot, the view, sensors and map settings you see in is loaded from the *view.vgz* configuration file.

Run your *bayes_filter* node (see Section 1), and then you can move your robot around with the controller.

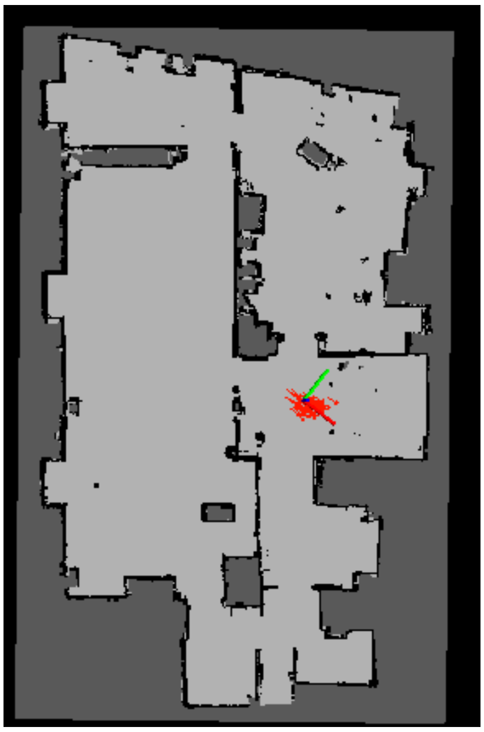
Make any optimizations you see fit.

Section 3 - Particle Filter - 20%

This section consists of implementing a particle filter for Monte Carlo localization, using ROS (Robot Operating System).

There already is a package in ROS for doing Monte Carlo localization: the *amcl* package. This section is intended as a replacement for AMCL.

Your task is to implement a class that inherits from the *MCLocalizer* interface and implements the missing methods. Specifically, you should implement a sensor model and a motion model, as well as the particle filter update rules. Skeleton code for running a simulator with a map and a robot, and publishing information from the particle filter, is available.



Installation:

In addition to the packages installed with the full desktop install, you also have to install *occupancy_grid_utils*. These are useful tools for dealing with grid maps, and are required for compiling the *MyLocalizer* class.

Install these packages (occupancy_grid_utils, particle_world, particle_filter) we created for you. Just unzip them in your *~/catkin/src*workspace directory.

1. Install an additional ROS package required to compile *occupancy_grid_utils*. Just type in a terminal:

```
sudo apt-get install ros-{your version of ROS}-tf2-bullet
```

2. Compile your catkin workspace. (if in doubt, see Section 1 how to do this)
3. There are two launch files available, which make it more convenient for you to run all components of ROS that are needed for the exercise. The launch files are called *particlefilter_lab.launch* and *particlefilter_willow.launch* are located in the *particle_world* package. The Lab map is a small and simple map; the Willow map is a much larger and more complex map. Launching these should open up two windows – Stage and RViz – both displaying the robot and a map.
4. In the Stage window, you can drag the display with the mouse, so that you can see the simulated robot. You can also enable display of the simulated laser range finder, under the menu "View / Data".
5. In the RViz window, make sure that the fixed frame is set to */map*.The RViz configuration file is also located in the *particle_world* package.
6. You can move the robot around with the *turtlebot_teleop* node or you can try to use your random walk algorithm you implemented in Section 1.
7. Run the provided MCL node (*particle_filter*). It does not do much yet, but it should show a set of particles as red arrows in the RViz window. The particles should move around as the robot moves.
8. Make any optimizations you see fit.

The section

The class *MyLocalizer*, located in *ParticleFilter.hpp*, and *ParticleFilter.cpp* has more-or-less empty methods for implementing a sensor model, a motion model, and a particle-filter update (weighting and resampling) routine.

Your task is to complete the predefined methods in *ParticleFilter.cpp* needed for localization.

More instructions and explanations are supplied in the source files.

Hints:

- Start with the Lab map, later try if you can localize yourself in the Willow map (you might need more particles).
- Use Gaussians for movement and measurement noise, you should figure out some sensible parameters for your noise.
- You can set your initiation location in RViz with the 2D Pose Estimate command.
- Raycasting is done for you (*simulatedScan*).
- Use a limited amount of rays (~30).
- Resampling -> resampling wheel.

Section 4 – 2D Navigation System - 20%

In this section you are going to build a 2D navigation system for a simulated robot using ROS (Robot Operating System).

Naturally navigation task is crucial for robot operation and ROS has a full scale navigation stack operated by *move_base* package. This section aims at replacing the existed stack with a more basic version.

You are provided with the known environment, configurations and a base code for the *Navigation* class you are going to contribute to.

Installation:

Install these packages (navigation_world and navigation) we created for you, you should just unzip them in your *~/catkin/src*workspace directory. After that:

1. Compile your catkin workspace. (see Section 1, how to do this)
2. There are two launch files available, which make it more convenient for you to run all components of ROS that are needed for the exercise. The launch files are called *lab.launch* and *willow.launch* are located in the *navigation_world* package. The Lab map is a simple map, the Willow map is a more complex map. Launching this should open up two windows displaying the robot and a map.
3. In order to try-out different starting positions you can use the *turtlebot_teleop* node (see Section 1 for reference).
4. Run the *navigation* node, which should just cleanly start and output you some info about the world it fetched.
5. Target point can be input to the *navigation* node in two ways, by:
   - adding command line arguments, i.e.

     ```
     rosrun navigation navigation tgtX tgtY
     ```

     , where tgtX and tgtY are your values;

   - using the "Publish Point" utility in RViz (just place the point anywhere on the map and your path-planning routine will get invoked).

The section

Navigation class, declared in *navigation.hpp* and specified in *navigation.hpp*, has more-or-less empty methods for path-finding and path visualization. The code provided gathers the all the required information about the world for you.

The source files contain the instructions and explanations needed to complete the section.

Requirements:

- Prepare the received knowledge about the world in a way to be later used by your path-planning algorithm.
- Perform path-planning, allowing the robot to plan its travels from any point A to any point B on the map.
- Get the path visualized in RViz.
- Make sure that the suggested plan correctly (dis-)allows the bot to travel through the narrow spots and go around the corners.
- Add functionality allowing the bot to actually travel from point A to point B, by following the path suggested. Ensure that the path execution is done without significant drift-off.
- (For higher marks) Try to optimize the path-planning algorithm computation time, suggested path length, and # of robot heading changes — in order to efficiently deal with the resources.
- (For higher marks) Extend the path-planning in such a way that a shortest path is found for an unordered set of N-path-points. Let the robot follow this more complex path.

Section 5 – Documentation – 30%

In this section you are asked to document the work you have done on this project

For each of the sections 2,3 & 4 answer the following questions.

1. Identify the AI algorithms used.
2. Describe with the aid of notation and diagrams how these AI algorithms are used.
3. Discuss the suitability of the simulated robot to perform the tasks
4. Explain the theory of any optimizations that you have used and comment on how the robot behaviour has been influenced.

Please include a section entitled "Division of Labour" containing the details below.

The section below must be cut and paste to the top of your documentation section.

Student Name1:	Student Number1:
Student Name2:	Student Number2:

Please complete the sections below with regard to the estimate of the division of work between the two partners

If the work was split in the range of 45% to 55% per partner, then that is fine and simply say "Work was evenly divided". If this was not the case, then state with a summary sentence. This is the important statement of this file.

Division of work: work was evenly divided

Code repository log (if applicable)
Paste here

Percentage of work completed by each partner on each class / task
Some areas require more work than others, so this is only for reference. An average of these values will not be calculated.

Filename / Task	**Student Name 1**	**Notes**	**Student Name 2**	**Notes**
Task1	40%		60%	
Task2	60%		40%	
Task3	Etc.			
Task5				
Task6				