FINAL

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2017-2018

BEng Honours Degree in Computing Part III
BEng Honours Degree in Electronic and Information Engineering Part III
MEng Honours Degree in Electronic and Information Engineering Part III
BEng Honours Degree in Mathematics and Computer Science Part III
MEng Honours Degree in Mathematics and Computer Science Part III
MEng Honours Degrees in Computing Part III
MSc in Computing Science (Specialist)
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Associateship of the City and Guilds of London Institute

PAPER C333

ROBOTICS

Friday 15 December 2017, 10:00
Duration: 120 minutes

Answer THREE questions

Paper contains 4 questions
Calculators not required

1 A robot has wheel odometry and a forward-facing sonar sensor, and holds a map which specifies the locations of the walls of a room relative to a standard coordinate system. In order to localise against this map as it moves, the robot is to perform Monte Carlo Localisation (MCL), using a cloud of weighted particles to represent a probabilistic estimate of its pose.

After some period of operation, its estimate is represented by `particleList`, a globally defined Python list of $N = 100$ `particle` objects. Each of these objects has data members `particle.x`, `particle.y`, `particle.theta`, and `particle.weight`. The particles' weights are normalised.

From this initial uncertain location, the robot is to drive towards a waypoint specified by coordinates W_x, W_y. It will achieve this by driving in steps, in each of which it first rotates on the spot to orient itself towards its best estimate of the current direction of the waypoint, then drives 20cm forwards, stops, makes a sonar measurement and updates its particle distribution to represent its new pose estimate. Write clear and precise Python-like pseudocode for the following elements, which together make up one of these steps.

a A function `CalculateMeanEstimate()` which returns `(xMean, yMean, thetaMean)`, the current mean estimate of the location of the robot. Assume that the particle distribution is fairly well clustered such that a simple mean is sufficient here.

b A function `DriveToWaypoint(Wx, Wy, xMean, yMean, thetaMean)` which makes appropriate calculations and then uses built in functions `RotateRobot(alpha)` and `DriveRobotForward(D)` to actually move the robot through one 20cm step. Use metre and radian units throughout, and assume that positive rotation is anticlockwise. This function should also return `(D, alpha)` to be used in the next part.

c A function `MotionPrediction(D, alpha)` which updates the particle set to represent the motion of the robot. Remember that the robot rotates, then drives forwards, and that it does not use any sonar sensing in between these actions. Make reasonable assumptions about the robot's motion uncertainty and include parameters which could be calibrated. You can use library function `random.gauss(mean, sigma)` to sample a single random number from a Gaussian distribution.

d A function `MeasurementUpdate(z)` which uses an unnormalised Gaussian likelihood function to alter the weights of all particles in response to a sonar depth measurement `z`. Python's library function `math.exp()` can be used for exponentiation. You can assume that a function `GetDistanceToWall(x,`

`y, theta)` is available which will calculate and return the forward distance to the closest wall from any robot location (x, y, θ) within the map.

e A function `void NormaliseParticleSet()` which normalises the particle set.

f A function `ResampleParticleSet()` which resamples the normalised particle set to replace it with particles which have equal weights but a spatial distribution representing the robot's position estimate. The library function `random.random()` returns a uniformly sampled random number in the range 0.0 to 1.0.

The six parts carry, respectively, 15%, 20%, 20%, 15%, 10%, and 20% of the marks.

2a A robot operates in area containing obstacles which have previously been accurately mapped. There are 10 obstacles in total, and each has a circular shape with radius 0.1m. The locations of the obstacles are available in `obstacles`, a global Python list of `obstacle` objects which store coordinates `obstacle.x` and `obstacle.y`. The robot has a circular shape with radius 0.15m, and its position and orientation is specified with standard coordinates (x, y, θ). Write a function `calculateObstacleDistance(x, y, theta)` in Python pseudocode which returns the distance between the boundary of the robot and the closest obstacle. The function should return zero if the robot is in contact with an obstacle, and a negative value if it is in collision.

b Using this function, write a full program which enables a differential drive robot to plan a safe path across the area from starting pose $(x, y, \theta) = (0, 0, 0)$ to a target location $(x, y) = (5.0, 0.0)$. The robot should use local planning similar to the Dynamic Window Approach. At an update rate of 10Hz, it should choose between nine options for the combination of the linear velocities v_L and v_R of its two wheels. Each of v_L and v_R can be kept the same, or increased or decreased by a single step of 0.05ms^{-1} in one timestep. Choose the best wheel velocities at each time step based on a suitable benefit term to reduce distance from the target location, and a cost term to keep the robot away from obstacles, with a look-ahead time $\tau = 1.0\text{s}$. The maximum positive or negative velocity of each wheel is 0.5ms^{-1}. The chosen new wheel velocities can be set with `setWheelVelocities(vL, vR)`. Assume that the robot moves precisely as commanded, and that it starts at rest. The general kinematics of a differential drive robot with width W, representing a pose update during motion along a circular arc during time Δt with wheel velocities v_L and v_R, are:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x + R(\sin(\Delta\theta + \theta) - \sin\theta) \\ y - R(\cos(\Delta\theta + \theta) - \cos\theta) \\ \theta + \Delta\theta \end{pmatrix},$$

where:

$$R = \frac{W(v_R + v_L)}{2(v_R - v_L)} \quad , \quad \Delta\theta = \frac{(v_R - v_L)\Delta t}{W}.$$

Remember to implement special simpler cases either when the robot drives in a straight line or makes a turn on the spot. The robot has width $W = 0.30\text{m}$.

c Briefly discuss the problems of a purely local planning approach with limited look-ahead, and explain the concept of global planning which can be used instead of or in combination with a local method.

The three parts carry, respectively, 20%, 60%, and 20% of the marks.

3 a A differential drive robot has two sonar sensors which face horizontally in the sideways left and right directions respectively relative to its forward movement direction. It must move forwards through a cave tunnel, which has unevenly shaped walls on both sides but is wider than the robot at all points. In Python-like pseudocode, implement a proportional control servo law which takes measurements from both sonars at a fixed rate of 10Hz and aims to steer the robot safely down a path in the middle of the tunnel by aiming to stay the same distance from both walls. The robot should move forward at a steady reference speed $v = 0.5\text{ms}^{-1}$ while adjusting the relative velocities of its left and right wheels in order to steer. It can set the linear velocities of its left and right wheels using the built-in function `setWheelVelocities(vL, vR)`, and it can obtain the current measurements from its sensors using `getSonarMeasurements()` which returns readings `(zL, zR)`. Note that the sideways-looking sensors are mounted slightly forward of the centre of the robot to make them more suitable for this servoing behaviour. Your program should include a suitable tunable gain constant. It should also use median filtering of the past 5 timesteps to ensure that occasional garbage measurements do not cause major problems.

b Once the controller above has been well implemented and tested, we can assume that the robot will smoothly move down the centre of the tunnel. At any point, the sum of its sonar measurements $z_W = z_R + z_L$ will be a measurement of the current width of the irregular tunnel. The robot records the complete history of these width measurements as it moves along the tunnel, which is 50m long in total, and saves them to a file at the end.

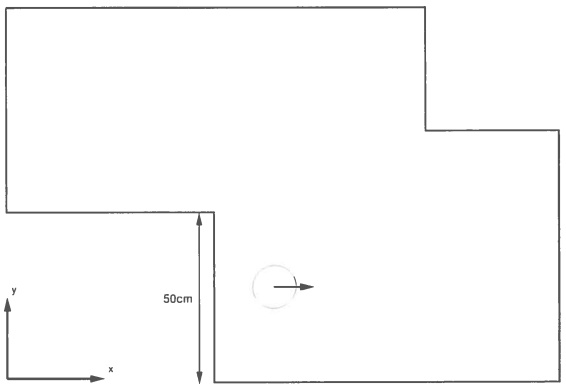
Explain with diagrams and formulae how the robot could use a signatures method to relocalise against this width measurement history if in the future it were to become lost in the tunnel. What would the robot need to do to determine its distance along the centre of the tunnel from the entrance, and with what accuracy would we expect it to be able to find this distance? Assume that the tunnel is irregular enough that any 1m of its length is unique in terms of width profile.

The two parts carry, respectively, 60%, and 40% of the marks.

4a A robot has an infra-red depth sensor with minimum range 50cm and maximum range 4m. The sensor is precise but not very robust at close range, with Gaussian-distributed zero-mean uncertainty of standard deviation 1cm at its minimum range when it receives a good measurement but on 24% of measurements it reports random 'garbage' values between the minimum and maximum range. At greater range the precision decreases such that the standard deviation of good measurements is proportional to the square of ground truth distance; but becomes more reliable, with a 'garbage' measurement rate inversely proportional to depth.

Draw a suitable likelihood function $p(z|m)$ for the sensor in the form of three separate 2D 'slice' plots which show the probability of obtaining depth measurement z at ground truth distance m =1m, 2m, 3m respectively. Your plots should be carefully drawn and annotated with the appropriate numerical values.

b A way for a different robot to find its location in a room without learning signatures in advance is to attempt global relocalisation against a map using Monte Carlo Localisation. Suppose that the robot is provided with the following map:



i) Make a copy of the diagram above, and draw 50 arrowed circles like the one shown to indicate how the robot would initialise a particle distribution to represent its initial state of knowledge that it is definitely within the room, but in an otherwise unknown position and orientation.

ii) Without moving, the robot now points its sonar sensor forward and makes a single measurement. It records a depth of 30cm, and executes the measurement update, normalisation and resampling steps of MCL to update its particle distribution. Make a new drawing of the mapped area, and copy into it only those particles from your previous drawing which would have been highly likely to have obtained large weights after the measurement and therefore be copied in the resampling step. Assume that

the sonar sensor has no systematic error, is robust with no 'garbage' measurements, and reports measurements with fixed standard deviation around 5cm.

iii) In its next full MCL iteration, the robot makes a precise turn on the spot of 90° to the right, makes another depth measurement, obtains a reading of 20cm, and then completes update, normalisation and resampling steps. Draw one more diagram, copying from your previous one any of the original particles which would still be likely to survive.

iv) Explain why and how the robot would be able to perform global relocalisation more efficiently if it also had a magnetic compass sensor.

The two parts carry, respectively, 40%, and 60% of the marks.