Cupp-E

The cup collecting robot

Participants	Kent Stark Olsen, B.Sc. student, Robot Systems Engineering The Maersk Mc-Kinney Moller Institute
	Lasse Simmelsgaard Børresen, B.Sc. student, Robot Systems Engineering The Maersk Mc-Kinney Moller Institute
	Leon Bonde Larsen, B.Sc. student, Robot Systems Engineering The Maersk Mc-Kinney Moller Institute
	Rudi Hansen, B.Sc. student, Robot Systems Engineering The Maersk Mc-Kinney Moller Institute

| Deadline | 2012-10-22 |

Faculty of Engineering
University of Southern Denmark
Niels Bohrs Allé 1
5230 Odense M
Denmark

www.sdu.dk/tek
+45 6550 7303
tek@tek.sdu.dk

Chapter 1

Summary

As data has been collected and empirically analysed by the canteen personal, found that researchers and students are lazy and will not bring their used service back to the canteen for cleaning. To accommodate this problem a robot is being designed which is able to collect "forgotten" cups and while performing this task also sweeps the floors. The robot platform has been developed but algorithms for the motion planning is still needed. This summary will give an overview of the software design for the robot.

Chapter 2

Coverage

2.1 Strategy

The coverage method used is boustrophedon, meaning “the way of the ox” from the plowing pattern of old days. The advantage of this strategy is that it ensures full coverage of a room with no obstacles and it is simple.

It is clear that when actually implemented on a robot, it must be supported by online sensors to cope with the obstacles present in the room, but for the purpose of estimating the distance travelled by such a robot, it gives a best case route.

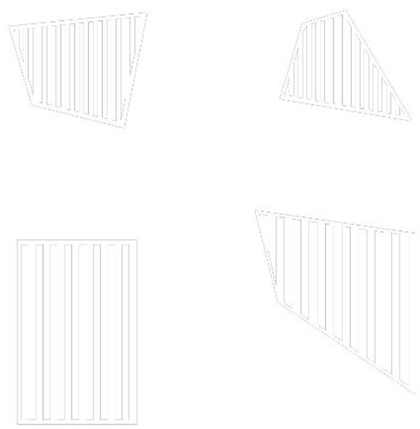

Figure 2.1: Shows example cells covered

2.1.1 Algorithm

The algorithm developed for covering follows the points listed in 1. The algorithm assumes the cell has been shrunk so the area given in the cell structure represents the free spae in which the robots center can move.

The algorithm first constructs a polygon from the cell data. Then the cell is swept by discrete vertical lines for every distance given by radius. This way a list is constructed holding the points where the sweeplines intersect the cell polygon, thus giving the robots turning points. Finally the list is sorted to reflect the order in which the turning points should be visited.

Algorithm 1 Boustrophedon coverage

```
Input:
cell                                   ▷ Structure holding vertices and edges
radius                                 ▷ The radius covered by the robot
list                                   ▷ empty list for returning the route
Output:
distance                               ▷ the lapsed distance
function COVER AREA(cell , radius , points)
    for each edge in cell do
        push line segment to polygon   ▷ This constructs a polygon shaped as the cell
    end for
    while sweepline inside polygon do
        construct next sweepline
        find intersections with polygon
        push intersections to points
    end while
    for each point in points do
        order points                   ▷ up,down,up,down sequence
    end for
    for each point in points do
        remove dublicates              ▷ in case intersection is a vertice
    end for
    for each point in points do
        calculate distance
    end for
    return distance
end function
```

2.1.2 Running time

The running time of this algoritm depends on the number of turningpoints in the returned path and theese again depends on the width of the cell and the radius of the robot.

$$n = 2 \cdot \frac{width}{radius} \tag{2.1}$$

Finding the point requires finding the possible intersections between a sweepline and the cell polygon. Since most cells are polygons of four vertices this means on average checking each sweepline against four lines or two checks per turning point. Checking the intersection between two lines means solving the equaled line equasions and is thus done in constant time giving this problem a complexity of O(n).

Ordering the list means iterating the list and reordering the points not fulfilling the criteria. Since swapping two entries of a vector is done in constant time, the operation is done in linear time.

Checking for dublicates means iterating the list and for each point checking the rest of the list for similar points. This operation is done i quadratic time.

Calculating the travelled distance is done by iterating through the list and adding up all the distances. Finding the distance between two points is done in constant time and thus the complexity of this problem is O(n).

2.1.3 Conclusion

Overall analysis of the algorithm thus shows that it runs in quadratic time due to the dublicate check and thus any effort to optimize the algorithm should start here, but since most cells are relatively small, the problem is also small.

CHAPTER 3

Cupscanner

The cup scanning algorithm was designed to take the workspace map (.pgm-file) and the point from which the robot would travel and the point to which it would travel. It returns a list of points where cups have been detected. Figure 3.1 illustrates a visualization of the algorithm and together with algorithm 2 a brief overview will be presented here.

3.1 Brief overview

The design of the algorithm had to solve the general problem of how to detect cups from a region determined by the shape of the robot and the two points in which the robot travels between. To solve this problem it is necessary to take into consideration that the robot can travel in any direction, and this can lead to divisions by zero if for instance $arctan$ is used to determine some angle of travelling, thus the algorithm is designed to not utilise mathematical tools which will lead to singularities. The algorithm is designed with help from mathematical vectors to create auxiliary points, and then lines and intersections are calculated with Boost Geometry. Figure 3.1 illustrates the geometries which is used to calculate this region of interest, this region is bounded by the bounding box and the red lines. Algorithm 2 explains in more detail what is going on behind the scene. The complexity of the algorithm is $O(N)$, where N is the numbers of points scanned between two points, N is dependent of length of travel and the radius of the robot. As the bounding region that is actually scanned does not match up with the ideal geometry of the robot an error is introduced, this error could possibly lead to cups are not detected or cups are detected without being in the actual range of the robot.

Algorithm 2 Cup scanner algorithm

Input:
map ▷ Workspace map where cups are represented
$from$ ▷ The point where the robot is located
to ▷ The point where the robot needs to go
$radius$ ▷ The radius covered by the robot
Output:
$listOfCups$ ▷ A list which contain position of cups detected

function CUPSBETWEENPOINTS(map, $from$, to, $radius$)
  create auxiliary points A, B, C, D, E, F, A_x, B_x, C_x, D_x from points $from$, to
  create bounding box $bBox$ from points A, B, C, D, E, F ▷ Creates area of interest
  create auxiliary line $lineA$ from points A_x, B_x
  create auxiliary line $lineB$ from points C_x, D_x
  create auxiliary line $travelLine$ from points E, F
  for each y in $bBox$ **do**
    clear $listOfIntersections$
    create new $scanLine$
    set minimum and maximum values of x from $bBox$
    calculate intersections between lines $scanLine$, $lineA$, $lineB$, $travelLine$
    if number of intersections equals 3 **then** ▷ Decides if $minX$ and/or $maxX$ needs adjustment
      decide which intersection point has lowest and highest x-value
    else if number of intersections equals 2 **then**
      decide if intersection with either line $lineA$ or $lineB$ creates a maximum or minimum value of x from the intersection with $travelLine$
    end if
    for each x between $minX$ and $maxX$ **do**
      if $map(x, y)$ is a cup **then**
        add point to $listOfCups$
      end if
    end for
  end for
  return $listOfCups$
end function

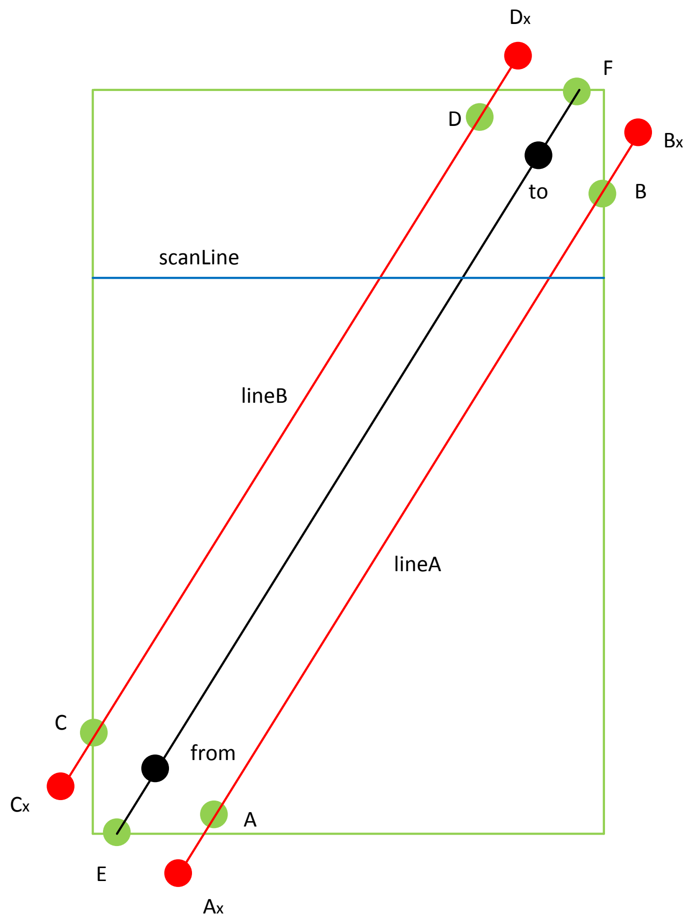

Figure 3.1: Illustrates how the the bounding box (green) and the red lines A and B defines the area of interest.

CHAPTER 4

Online planning

4.1 Simulation

The simulation is done online, but seeks to reduce the output to a minimum. The main task is to visit all cells in the map and sweep them. This is simulated as an online statemachine changing between different brahaviours of the robot.

4.1.1 Behaviours

There are basically three beahaviours of the robot:

- Select a destination
- Go to a destination
- Cover a cell

The main task is solved using theese three behaviours encapsulated in a state machine. The state machine concists of two parts. The first part is a decision maker that selects a route and the other is an onlie stepper to follow the route.

4.1.2 The decision maker

The decision maker selects a route based on the current state and is thus a state machine. In either state it facilitates the behaviours to select a route implemented as a list of points to visit followed by a state transition based on state variables. The state machine follows the diagram in figure 4.1.

4.1.3 The stepper

The stepper is a sequential piece of code that iterates through the list of points adding up the travelled distance, checking for cups in range, collects the cups and returns the cups when tray is full. The task of returning the cups only contributes to the travelled distance since the robot cannot collect cups and the concept is therefore implemented as a mere calculation. A flow chart for the stepper can be seen in figure 4.2

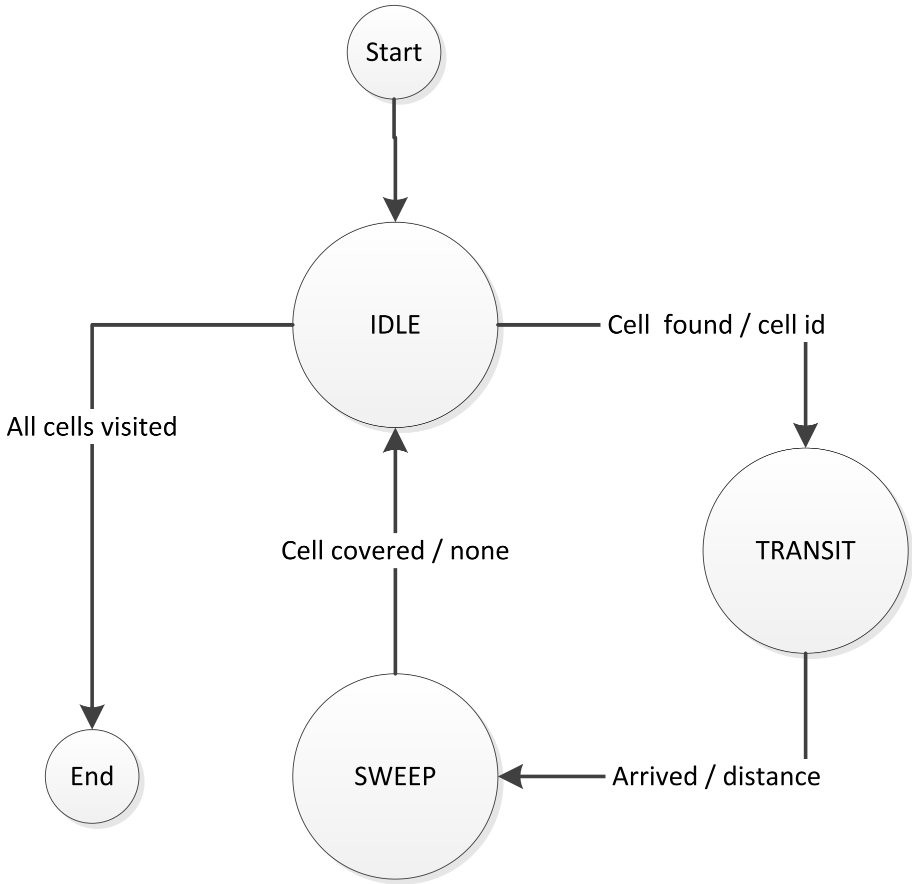

Figure 4.1: State machine diagram of the decision maker

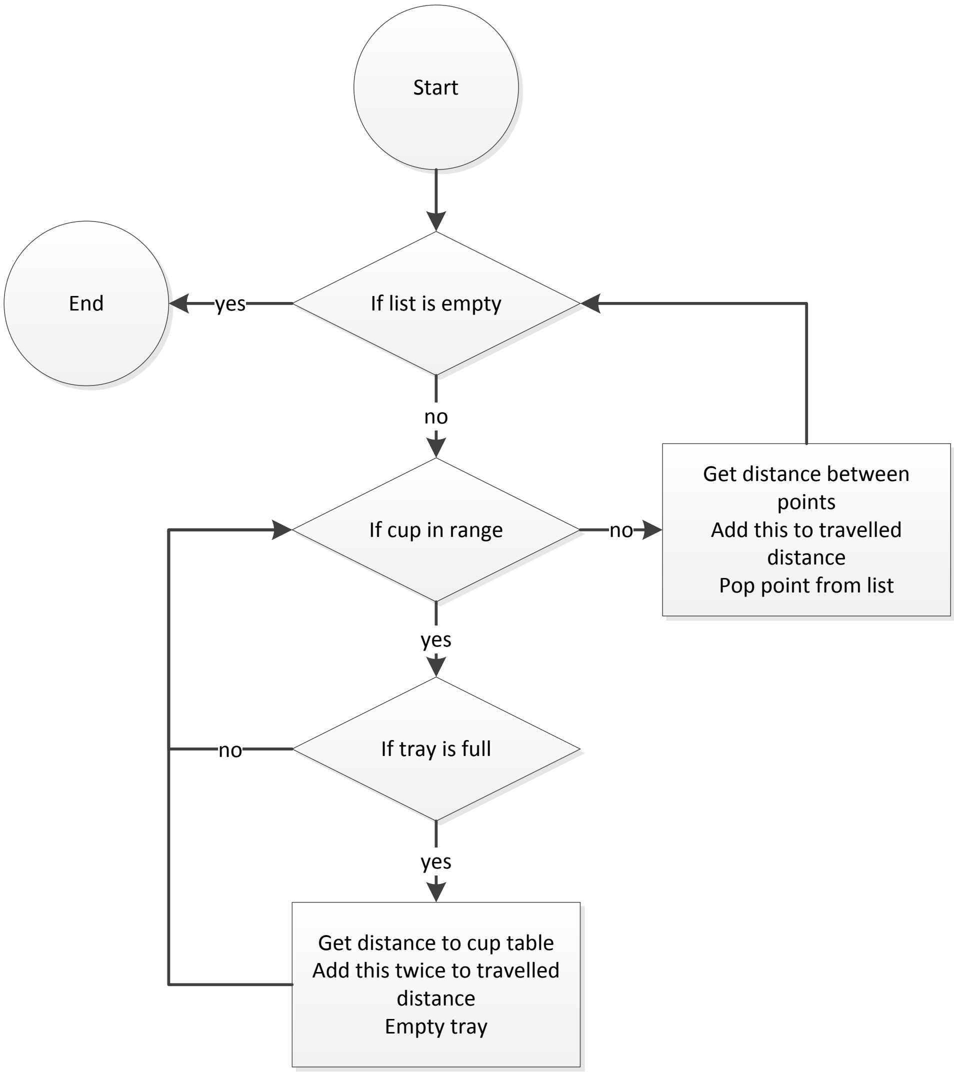

Figure 4.2: Flow chart of the stepper