Path Planning Algorithms For The Robot Operating System

Aleksandar Tomović
Computer Science
Saint Cloud State University
Saint Cloud, MN 56301- 4498
toal1201@stcloudstate.edu

Abstract

The open-source Robot Operating System (ROS) is a heterogeneous and scalable P2P network-based robotics framework. We present path-planning algorithms over a P2P network for collision-free autonomous ground-robot navigation under uncertainty constraints. We then describe a practical application of this navigation facility to a natural stone processing plant factory floor.

1. Introduction

The ROS robotics *framework*, including affiliated open source autonomy libraries, all integrated into a ground robot equipped with sensors, (essentially called a work platform), is sufficient to present robot behaviors, robot manipulation, and robot (P2P) communication as part of our research conducted in the plant. With basic robotic knowledge now we realize how quickly robots may be used in an industrial environment, to help improve production flow and cost efficiency. This paper presents an easy implementation of robots into a natural stone production process. Robotic technology connects information to the physical world around us. Such applications include: unmanned ground vehicles (UGV), unmanned aerial vehicles (UAV), and robots exploiting visual optics that can use the internet to navigate with partial classification. A robot's structure, chassis, geometry, and joints, are all important aspects of robot design. The knowledge of kinematics and dynamics (how a robot moves), is essential for a successful design. During our research we adapted an already developed platform available on the market, to perform production tasks. The ROS is supported by full time developers from the Open Source Robotics Foundation (OSFR) and by independent contributors including graduate students, robot builders, application developers, and hobbyists [11][1]. This paper is structured as follows: Section 2 discusses the robot communications, Peer-to-Peer network and networks styles of communication. Section 3 explains how to develop an intelligent robot considering navigation and paths planning algorithms. Section 4 discusses path planning with uncertainty, tools, and nonlinear state estimation. Section 5 presents the navigation problem of autonomous robots, simultaneous Localization and Mapping. Application algorithm in natural stone processing plant is described in Section 6. Conclusion is outlined in Section 7.

2. Peer-to-Peer Network

A P2P network is a decentralized and distributive network architecture in which all individual nodes are called "peers". In decentralized P2P network peers arrange themselves into an *overlay network* which is a logical network made on top of a physical network. Nodes are processes performing computation in the ROS system (*Vertices* in the ROS computational graph). Each *instance* must have a unique name and type (filesystem location of the executables). All the messages are *packets* of data sent between ROS nodes. Unidirectional communication links between ROS nodes is named *topic.* ROS is a distributed computing environment comprised of hundreds of nodes and multiple machines. Any node may communicate with another at any time. ROS P2P network is loosely coupled and uses a few different styles of communication.

The following paragraphs describe *synchronous* RPC communication, *asynchronous* streaming and parameter server.

Synchronous RPC communication over services: The RPC protocol allows the construction of client-server applications, using a demand/response protocol with management of transactions. The client is blocked until a response is returned from the server, or a user-defined optional

[1] Contributions are coordinated via `ROS.org`, compiling wiki documentation, and the ROS standards [11].

timeout occurs. RPC guarantees at-most-once semantics for the delivery of the request. It also guarantees that the response received by a client is definitely that of the server and corresponds effectively to the request (and not to a former request to which the response might have been lost). RPC also allows a client to be unblocked (with an error result) if the server is unreachable or if the server has crashed before emitting a response. Finally, this protocol supports the propagation of abortion through the RPC. This mechanism is called *abort propagation*. When a thread that is waiting for an RPC reply is aborted, this event is propagated to the thread that is currently servicing the client request.

Asynchronous streaming of data over topics: *Asynchronous transmission* uses start and stop bits to signify the beginning and ending bits, so a character would actually be transmitted using ten bits instead of 8. For example, "0100 0001" would become "**1** 0100 0001 **0**". The extra one (or zero, depending on the parity bit) at the start and end of the transmission tells the receiver first that a character is coming and secondly that the character has ended. This method of transmission is used when data are sent intermittently as opposed to in a solid stream. In the previous example the start and stop bits are in bold. The start and stop bits must be of opposite polarity. This allows the receiver to recognize when the second packet of information is being sent.

Storage of data on *parameter server*; A parameter server is a shared dictionary that is accessible via network APIs. Nodes use this server to store and retrieve parameters at runtime. As it is not designed for high-performance, it is best used for static, non-binary data such as configuration parameters. It is meant to be globally viewable so that tools can easily inspect the configuration state of the system and modify if necessary. The parameter server runs inside of the ROS master, which means that its API is accessible via normal RPC libraries [11].

.

3. Robotic Navigation

Developing *intelligent* robots that can accomplish production tasks without human help has fascinated many in artificial intelligence communities. From a technical point of view, autonomous robot navigation focuses primarily on generating optimal global paths to maneuver the robot to a target production station in a real time environment. Autonomous robots can reactively correct their course by circumventing obstacles with collision avoidance and at the same time explore and map the unmapped region. When the robot sees a target from the distance, it may not be an accurate distance. Using parameter estimation and a Kalman Filter [9] the robot can estimate the target position most accurately. Using all the data the sensors have processed, the robot uses a Kalman Filter for the parameter estimation and the state estimation. Figure 1 illustrate measurement uncertainty associated with the result.

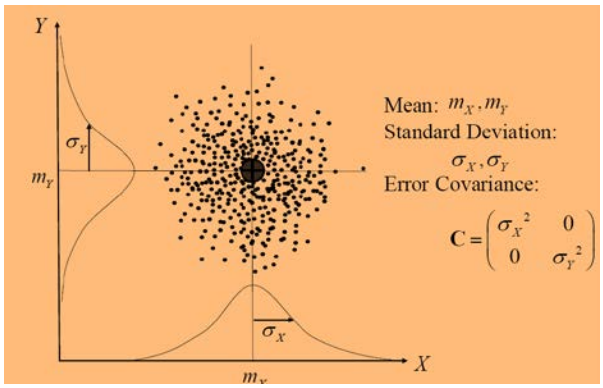

Figure 1: Quantifying Uncertainty [1]

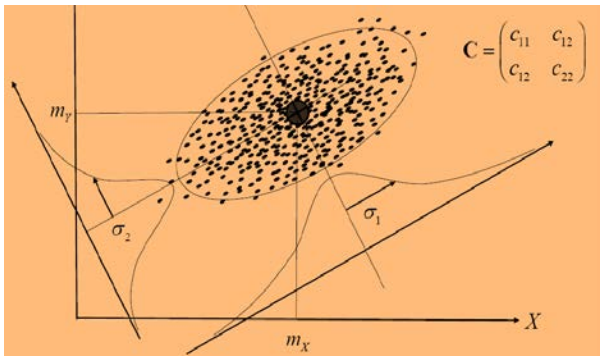

Figure 2: Covariance matrix [1]

Figure 2 illustrates a covariance matrix which contains off-diagonal elements, reflecting correlation between two axes.

Global path planning (GPP) addresses autonomous robot navigation in contexts including unmanned ground vehicles control [18], an unmanned aircraft [13], and the Mars Rover [15]. A *robot global path planning* (RGPP) system senses the information from the environment and plans a collision-free trajectory to navigate to a destination; in our case, the production station. Initially, GPP algorithms assumed a robot has complete knowledge of its environment, production floor, and its own placement. However, in real applications, information is partially available or completely unavailable in advance. Therefore, more recent work has focused on how to generate a global path in the presence of sensor noise and map incompleteness [12]. For GPP the paper used Dijkstra algorithm expressed in pseudo code, Figure 3 [16].

```
 1 Function Dijkstra (Graph, source):
 2      for each vertex v in Graph:          // Initializations
 3          dist[v]:= infinity;              // Unknown distance function from
 4                                           // source to v
 5          previous[v]:= undefined;         // Previous node in optimal path
 6      end for                              // from source
 7
 8      dist [source]:= 0;                   // Distance from source to source
 9      Q: = the set of all nodes in Graph; // All nodes in the graph are
10                                           // unoptimized – thus are in Q
11      while Q is not empty:                // the main loop
12          u: = vertex in Q with smallest distance in dist []; // Source node in
first case
13          remove u from Q ;
14          if dist[u] = infinity:
15              break;                        // all remaining vertices are
16          end if                            // inaccessible from source
17
18          for each neighbor v of u:         // where v has not yet been
19                                            // removed from Q.
20              alt: = dist[u] + dist_between (u, v);
21              if alt < dist[v]:              // Relax (u, v, a)
22                  dist[v]:= alt;
23                  previous[v]:= u;
24                  decrease-key v in Q;      // Reorder v in the Queue
25              end if
26          end for
27      end while
28      return dist;
29  end function
```

Figure 3: Dijkstra's pseudo code [16]

Figure 5 shows the A* algorithm which combine both Dijkstra and Best First Search presented in figure 6, to find the shortest path .The map navigation used ROS's move_base planner architecture. The default global planner is a wrapper around Dijkstra's algorithm. The default local planner is called *TrajectoryPlannerROS*. It wraps an approach called dynamic window.

```
1 OPEN = [initial state]
2 CLOSED = []
3 While OPEN is not empty
4 Do
         Remove the best node from OPEN, call it n, add it to CLOSED.
         If n is the goal state, backtrack path to n (through recorded parents)
         and return path.
         Create n's successors.
         For each successor do:
          a. If it is not in CLOSED and it is not in OPEN: evaluate it, add it to
                 OPEN, and record its parent.
          b. Otherwise, if this new path is better than previous one, change its
                 recorded  parent.
                 i.  If it is not in OPEN add it to OPEN.
                 ii. Otherwise, adjust its priority in OPEN using this new
                     evaluation.
5 Done
```

Figure 4: Best First Search Algorithm [16]

```
1 Function A*(start, goal)
2    closedset: = the empty set    // the set of nodes already evaluated.
3    openset: = {start}      // the set of tentative nodes to be evaluated,
                             //initially containing the start node
4    came_from:= the empty map    // the map of navigated nodes.
5    g_score [start]:= 0    // Cost from start along best known path.
6    // Estimated total cost from start to goal through y.
7    f_score [start]:= g_score [start] + heuristic_cost_estimate (start, goal)
8     while openset is not empty
9        current: = the node in openset having the lowest f_score [] value
10        if current = goal
11          return reconstruct_path (came_from, goal)
12          remove current from openset
13          add current to closedset
14        for each neighbor in neighbor_nodes (current)
15          if neighbor in closedset
16            continue
17          tentative_g_score:= g_score [current]+ dist_between (current,neighbor)
18          if neighbor not in openset or tentative_g_score < g_score [neighbor]
19              came_from [neighbor]:= current
20                g_score [neighbor]:= tentative_g_score
21                f_score [neighbor]:= g_score [neighbor]+ heuristic_cost_estimate
                                      (neighbor, goal)
22          if neighbor not in openset
23                 add neighbor to openset
24                 return failure
25   Function reconstruct_path (came_from, current_node)
26   if current_node in came_from
27        p: = reconstruct_path (came_from, came_from[current_node])
28        return (p + current_node)
29    else
30        return current_node
```

Figure 5: A* Pseudo Code [16]

Classical path planning (CPP) usually represents the world called the configuration space [18] [4]. The configuration space uses generalized coordinates (parameters that define the configuration of a system) and vector space. Methods which transform the configuration space into cell regions that can be used for path planning are termed cell decomposition methods. Cell decomposition (CD) methods are the most popular approaches to a wide number of applications in robotics. One of these methods, tiling the configuration space into convex polygons and termed cells, uses path search methods to search through cells to find the optimal path to the goal. *Roadmap methods* fill the configuration space with roadmaps/graphs that contain nodes representing reachable robot configurations and edges which are one-dimensional curves representing the free space between the nodes corresponding to topographical properties. Conventional Roadmap Methods for finding *shortest paths* include visibility graphs, which connect the nodes of polygonal obstacles, and Voronoi roadmaps which use borders as edges [5].Other methods in use are Probabilistic Roadmap Methods (PRM) (see Figure 6), Potential Field Methods (PFM), and Harmonic Potential Field Map Methods (HPFM). Robotic systems can achieve global planning and avoid being trapped in local minima. This is achieved by integrating the methods of certainty grids used for obstacle representation and potential fields devised for navigation and by considering the entire path [5].

```
 1  INITIALIZE()
 2  for all (cell ∈ cells) do
 3      cell.dist = distPointQueryLine(cell.origin);
 4   end for
 5   OP EN = {parentCell(n_init )}; 6:  CLOSED = ∅;
 7   GROWPRMINCELL(cell)
 8   numSamples = 0;
 9   while (numSamples < nodeIncrementPerCell) do
10          sample random configuration c_new in cell;
11      cell.numTrials++;
12      if (isFreeConfig(c_new)) then
13           add node n_new  to N ;
14           connect n_new  to neighbors, add edges to E;
15          update cell.numComponents of cell;
16           numSamples++;
17       end if
18   end while
19   cell.numSamples += numSamples; 20:  updateOccupancy(cell);
21   updateConnectedness(cell);
22   updateValue(cell);
23   SOLVEQUERY(n_init ,n_goal ) 24:  Initialize();
25   add n_init , n_goal  to N ;
26   connect n_init , n_goal  to neighbors, add edges to E; 27:  loop
28      if (OP EN = ∅) then
29           report failure;
30       end if
31       cell = takeFirst(OP EN );
32       GrowPRMInCell(cell);
33      PerformRandomWalksInCell(cell);
34      if (cell.occupancy > occupancyThresh or cell.numSamples ≥
                             maxNodesPerCell) then
35           CLOSED ← cell;
36       else
37           OP EN ← cell;
38       end if
39      for all (neighbor ∈ neighbors(cell)) do
40           if (neighbor ∉ CLOSED) then
41                OP EN ← neighbor;
42       end if
43       end for
44      sortAscendingByValue(OP EN );
45       if (parentComp(n_init ) = parentComp(n_goal)) then
46           path = findShortestPath(G);
47          if (path satisfies quality condition) then
48               return  path;
49           else
50               publish path;
51           end if
52       end if
53  end loop
54   MAIN()
55  create array cells; 56:  N = ∅;
57  E = ∅;
58  G = (N, E);
59  for all (query ∈ queries) do
60       solutionPath = SolveQuery(query.n_init,query.n_goal);
```

Figure 6: Cell Probabilistic RoadMap Method (PRM) pseudo code [19]

4. Path Planning with Uncertainty

The current position estimate of the robot is given by calculating the mean of the previous paths. However in practice, the estimate is likely to be noisy and we have to take this uncertainty into account in order to ensure collision free and efficient paths. The CPP is

performed by robots in fully observable environments. System conditions are usually assumed to be deterministic and discrete. Probability based approaches aided by statistical tools (extended Kalman Filter [9] and particle filter–sequential Monte Carlo [10]) were employed. The Extended Kalman Filter (EKF), figure 7, which is the nonlinear variant of the Kaplan filter linearizes about an estimate of the current mean and covariance. The EKF has been considered the standard in the theory of nonlinear state estimation, navigation systems, and GPS.

	Standard Kalman Filter	Extended Kalman Filter
	Linear systems	Nonlinear systems
State Equation	$x_{t+1} = A_t x_t + B_t u_t + G_t w_t$	$x_{t+1} = f(x_t, u_t, t) + G_t w_t$
Output Equation	$y_t = H_t x_t + v_t$	$y_t = h(x_t, t) + v_t$

Figure 7: Extended Kalman Filter (EKF) deals with nonlinear systems represented by nonlinear state and output equations [9], [1]

Suppose that robot is at point *A* at the time *t* having a state estimation error covariance *Pt*. The robot wants to acquire more data to estimate the location of its position relative to the target. Quantified gathering of new data provides the robot with more useful information (see Figure 8).

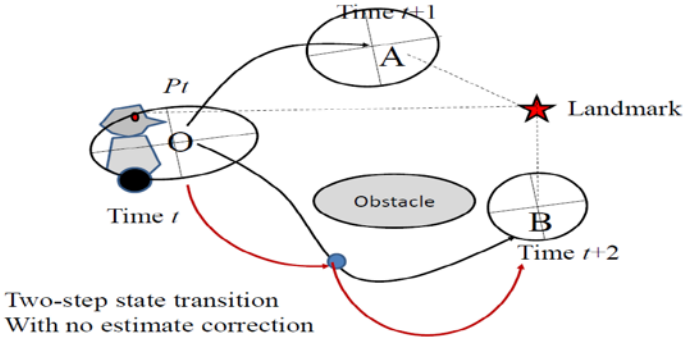

Figure 8 : Two-step state transition with no estimate correction [1]

The *particle filters* or *sequential Monte Carlo* (SMC) [10] method consists of a set of on-line posterior density estimation algorithms that estimate the posterior density of the state-space by directly implementing the Bayesian recursion equations. SMC methods use a grid-based approach, and use a set of particles to represent the posterior density. SMC methods provide a methodology for generating samples from the required distribution without requiring assumptions about the state-space model or the state distributions [10]. Figure 9 gives a map of the environment; the goal of the algorithm is for the robot to determine its pose within the environment.

```
1 Function MCL:     ( Xt-1, ut,zt)
2         Xt = Xt = 0
3                   for m = 1 to N
4                             Xt(m) = motion_update(ut,xt-1(n))
5                             Wt(m) = sensor_update(zt,xt-1(n))
6                             Xt = Xt +{Xt(n),wt(n)}
7                   end for
8                   for m  = 1 to M
9                   draw x_t^[i] from X̄_t with probability ∝ w_t^[i]
10                            Xt = Xt+ Xt(i)
11                  end for
12        return Xt
```

Figure 9: Particle Filter-sequential Monte Carlo Algorithm [14]

The basic MCL algorithm, pseudo-code for generating time t in the next set of samples S_{t+1} from current set S_t is illustrated in figure 10. The next x_t is the location and the w_t are the probabilities (x_t, wt) pair represents the sample. The distance traveled is u_t and the sensor reading is $z_{t.}$. The location of sample i at the time t is $x_t^{(i)}$,where n is number of samples.

```
1 Inputs: Distance ut, sensor reading zt, sample set St ={(xt(i), wt(i) )| i= 1…n}
2       for  i=1 to n do                  // first update of current set of samples
3               xt = updateDist(xt,ut)     // compute new location
4               wt(i) =prob(zt|xt(i))      // compute new probability
5       St+1 = null                       // resample for next generation of samples
6       for I = 1 to n do
7               Sample an index j from distribution given in weights St
8       Add (xt(i), wt(i)) to St+1// Add sample j to new set of samples
9       return St+1
```

Figure 10: MCL Basic Algorithm [14]

5. Simultaneous Localization and Mapping (SLAM)

During the process of path planning, the robot is continuously learning its environment. Presently performed by building a map of the environment where the robot is locating and localizing itself concurrently ,termed Simultaneous Localization and Mapping (SLAM). As the robot learns the locations, it will reach the destination more quickly and efficiently. The SLAM was introduced by IEEE Robotics during an automation conference [5]. Mathematically, SLAM, being a nonlinear filter, estimates the distribution over robot poses and significant locations in a recursive fashion over time, given sensor readings corrupted by noise and systematic errors. Slam shown in figure 11 maintains probability distribution over *current* pose and map (step 1), then predict time (step 2), and finally update the measure (step 3).

1. $p\ (x_{t+1}, M \mid z^{t+1}, u^{t+1})$,
2. $p\ (x_t, M \mid z^t, u^t)\ u_{t+1} \rightarrow p(x_{t+1}, M \mid z^t, u^{t+1})$,
3. $p\ (x_{t+1}, M \mid z^t, u^{t+1})\ \ z_{t+1} \rightarrow p(x_{t+1}, M \mid z^{t+1}, u^{t+1})$,

Figure 11: Probability distribution over current pose

Local navigation follows the *global path* and determines the next motion command based on local observations in real time. Local navigation generates a new path by overwriting the original global path in response to a regional change in an environment such as new obstacles. SPOTT's local path planner is based on a potential field method using harmonic functions, which are guaranteed to have no spurious local minima [17]. A harmonic function on a domain Ω C R^n is function which satisfies Laplace's equation:

$$V^2\phi = \Sigma_{i=1}^{n} \frac{\delta^2\phi}{\delta^2 x_i^2} = 0$$

The value of ϕ is given on a closed domain Ω in the configuration space C.
The default local planner is called *TrajectoryPlannerROS*. It wraps an approach called *dynamic window*. The DWA is a velocity-based local planner that calculates the optimal collision-free ('admissible') velocity for a robot required to reach its goal. It translates a Cartesian goal (x, y) into a velocity (v, w) command for a mobile robot. There are two main goals, calculate a valid velocity search space, and select the optimal velocity. The search space is constructed from the set of velocities which produce a safe trajectory i.e. allow the robot to stop before colliding, given the set of velocities the robot can achieve in the next time slice given its dynamics '*dynamic window*'. The optimal velocity is selected to maximize the robots clearance, maximize the velocity and obtain the heading closest to the goal (See Figure 12).

```
1 BEGIN DWA (robotPose, robotGoal,robotModel)
2   desiredV = calculateV(robotPose,robotGoal)
3   laserscan = readScanner()
4   allowable_v = generateWindow(robotV, robotModel)
5   allowable_w = generateWindow (robotW, robotModel)
6   for each v in allowable_v
7     for each w in allowable_w
8     dist = find_dist(v,w,laserscan,robotModel)
9     breakDist = calculateBreakingDistance(v)
10    if (dist > breakDist)  //can stop in time
11        heading = hDiff(robotPose,goalPose, v,w)
12        clearance = (dist-breakDist)/(dmax - breakDist)
13        cost = costFunction(heading,clearance, abs (desired_v - v))
14       if (cost > optimal)
15          best_v = v
16            best_w = w
17            optimal = cost
18    set robot trajectory to best_v, best_w
19  END
```

Figure 12: Local Planner Algorithm DWA

6. Robot Application in Natural Stone Processing Plant

Natural stone processing plants process stone pieces of different sizes and weights, which involve heavy lifting and manual labor. Autonomous robots may improve the production process, reduce production costs and enhance safety by introducing *robotic navigation* into the plants. Modern stone processing plants have sophisticated cutting equipment, but product

movement is labor-dependent and a major part of the production cost. Figure 13 illustrate how robots may complement to the "*U*" production cells [2] that are designed to reduce operator and product movement based on "*one pieces flow*" factory floor [3] illustrated in figure 2. With access to the ROS, industry is in a good position to implement robots into the production process for heavy lifting, transport, and safe waste removal, without the prohibitively high costs of robots in the past. Cells designed to eliminate waste help optimize material, people, and information flow. U, J, or L-shaped cells, with stations interlinked by manual roller conveyors, eliminate wasted space so operators can move swiftly from station to station without unnecessary steps or energy. Ergonomic principles can minimize reach distances and times to help eliminate worker fatigue. Parts should be provided from the rear of the cell via parts "stores". This simplifies replenishment and any line changeovers.

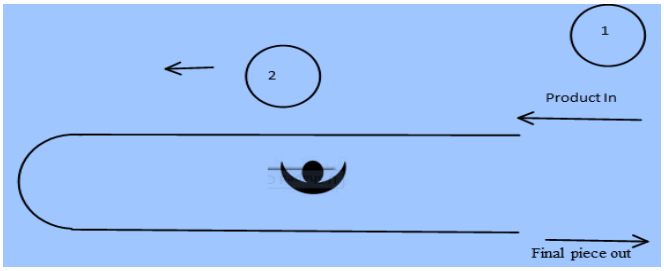

Figure 13: *"U"* Production in a Single Cell Robot Path.

One piece flow or *one piece at the time*, a product moves from machine to machine in the *"U"* shaped cells with a worker operating inside the *"U"* shaped cell and the robot moving parts outside following an also *"U"* shaped path. Figure 14 present production floors consist of multiple mobile *"U"* shaped cells spread out over the factory floor. Product movement, even some production steps such as precise stone cutting and waste removal, may now be done by the ROS autonomous robots. The key to optimizing material and people flow is to insist on one-piece flow making one complete part at a time, or passing completed work to the next operator only when that operator is ready for it. In a poorly balanced cell, work-in-process (WIP) stacks up between each station. This is waste. And quality re-works means you have to find the error and re-make a lot of inventory. With one-piece flow, you will find errors immediately and fix the problem. The robot path algorithm is developed to follow a *"U"* shaped line carrying one piece at a time from one machine station to the next. This approach allows us to divide the process into simple production stations. All robots communicate to each other to avoid collisions. Once one production process is completed robots move parts to the next station, or turn 180 degrees preparing the piece for another cut. The operator is situated inside the *"U"* shaped production cell, executing production tasks without extensive movements. Robot height is designed to provide a safe working environment for operators to avoid extensive bending. Product handling and heavy lifting is completely eliminated. The piece is completed once the robot with the piece leaves the production cell, and soon after the robot will decide where to take the piece next; either the next *"U"* shaped cell for another operation, or to the shipping warehouse.

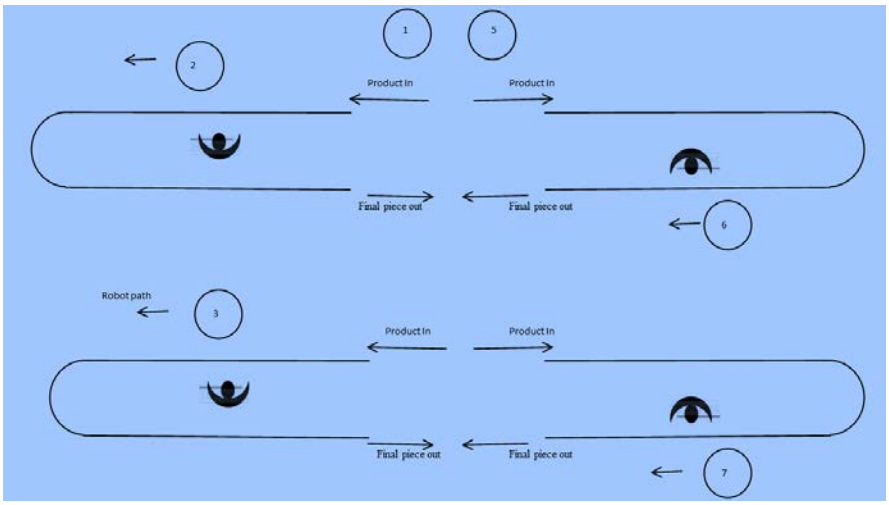

Figure 2: One Piece Production Flow with four *"U"* Shaped Spaces and Independent Production Cells

7. Conclusion

Real world robot navigations have significant challenges such as path planning, obstacle avoidance, fault isolation, and system resiliance. Autonomous robot movement (navigation) is a collection of hybrid design systems which are capable of handling all aspects of robotic navigation requirement. This paper presented paths algorithms used in the ROS. The open source library maintained by the robot volunteer community has proven a useful and helpful tool for all educational purposes. First, the paper presented robot movements in a fully functioned *one piece flow* production environment, using *"U"* shaped production cells for the robot's path. Thereafter we briefly described the robot processing plant implementation and robot movements on the factory floor. The *probabilistic* techniques are promising. Global planning, frameworks capable of coping with uncertainty have become increasingly popular. Partially Observable Markov Decision Process (POMDP), [7], and SLAM are advances in solving problems in global planning, local navigation, and exploration. The robot's local movement has a number of methods proposed for obstacle avoidance and horizon control. Research in autonomous robot navigation has made significant progress. Navigation applications require the capability to solve problems offering nearly optimal solutions. For this reason, global planning algorithms, local navigation routines, and exploration procedures must be integrated to achieve a global goal.

References

[1] MIT -Robotic Summer Course 2013 - Harry Asada - Ford Professor of Engineering, MIT, Michael Boulet a member of the technical staff in the Control Systems Engineering Group at MIT Lincoln Laboratory.
[2] http://www.tpslean.com/glossary/cellmfgdef.htm
[3] http://www.thetoyotasystem.com/lean_concepts/one_piece_flow.php
[4] Gehrig, S. & Stein, F. (1993). Collision avoidance for vehicle-following systems. IEEE Transactions on Intelligent Transportation Systems, 8(2), 233 – 244.
[5] Durrant-Whyte, H. & Bailey, T. (2006). Simultaneous localization and mapping: Part I. IEEE Robotics and Automation Magazine,13(2), 99–110
[6] Voronoi, Georgy (1908). "Nouvelles applications des paramètres continus à la théorie des formes quadratiques". *Journal für die Reine und Angewandte Mathematik*
[7] Smallwood, R.D., Sondik, E.J. (1973). "The optimal control of partially observable Markov decision processes over a finite horizon". *Operations Research* 21 (5): 1071–88
[8] Introduction to Modern Robotics II , edited by Daisuke Chugo, Sho Yokota, Concept press Ltd.
[9] R.E. Kalman (1960). "Contributions to the theory of optimal control". *Bol. Soc. Mat. Mexicana*: 102–119.
[10] Cappe, O.; Moulines, E.; Ryden, T. (2005).*Inference in Hidden Markov Models*. Springer
[11] http://www.ros.org
[12] http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6712082&tag=1
[13] Dalamagkidis, K., Valavanis, K. P., & Piegl, L. A. (2010). Autonomous autorotation of unmanned rotorcraft using nonlinear model
[14] http://en.wikipedia.org/wiki/Monte_Carlo_localization
[15] Tompkins, P., Stentz, A., & Wettergreen, D. (2004). Global path planning for mars rover
[16] http://en.wikipedia.org/wiki/
[17] Connoly C. and Grupen R, (1993). The Aplication of harmonic functions to robotics. Journal of Robotic Systems, 10(7);931-946,October
[18] Giesbrecht, J. (2004). The global path planning for unmanned ground vehicles. The technical report, exploration. In Proceedings of the IEEE Aerospace Conference - predictive control. Journal of Intelligent and Robotic Systems, 57(1-4), 351–369.
[19] http://www.lsr.ei.tum.de/fileadmin/backup/klasing-Cell-basedProbabilisticRoadmaps_ICAR2007.pdf