



Electrical & Computer Engineering Department

*ECE 7220 – Real Time Embedded Computing*

*ECE 7330 – Introduction to Mechatronic & Robotic Vision*

## **Semester Project**

*“Controlling Large Populations of Simple Robots  
with a Common Input Command”*

### Project Report

*Student Name:* **Giannis Fikas**

*PawPrint:* **jeff2b**

*Due Date:* **12/13/2013**

## A. ABSTRACT

The main objective of this semester project is to investigate controlling large populations of simple robots that each receives exactly the *same* input commands. Previous work has focused on approximately steering such a collection of robots to arbitrary *Cartesian* positions. The algorithm that is implemented in this project steers a robot population exactly to arbitrary *range* and *bearing* locations relative to targets in a *finite* number of steps. Also a *Real Time Database* is being used in the core of the system in order for the desired communication and synchronization to be achieved. Finally, to evaluate and visualize the performance of the implemented algorithm a *mobile robot* simulator and API are utilized.

## B. INTRODUCTION

It is commonly accepted that it is now possible to create and field very large populations of simple robots. This new approach would never be a reality if *micro-* and *nanorobots* couldn't be produced with an astonishingly small marginal cost. An increasing number of scientific fields embrace this approach to tackle problems that weren't possible to deal with before. Some important examples of such fields include *Biology* and *Chemistry* and some featured applications can be found in *targeted therapy, sensing* and *actuation*.

Even though manipulating a large population of molecular robots provides a powerful technique to deal with a lot of problems, it does not come without its own challenges. As the number of robots increases, some very important issues rise and demand attention. One such challenge is *how to perform state estimation* for these robots. This problem does not only get more challenging as the number of robots increases, but also as the robot size decreases. For example, at the molecular scale the number of individual modifications that can be made is limited.

Another important challenge that arises is *how to control the robots*. Up until recently, traditional approaches implement independent control signals for each robot of the population. It can be easily understood that this approach has some significant problems, as it requires a constantly increasing number of engineering and communications bandwidth. More recently, robots have been constructed in heterogeneity way so that they respond differently to a *common input signal*. Examples include these systems [1] – [4], which all rely on global inputs and each robot receives an exact copy of the control signal.

In this project, a *finite-time open-loop* algorithm is implemented to control a collection of robots that have *unique* turning rates. This algorithm provides the ability to steer a robot population exactly to arbitrary *range* and *bearing* locations

relative to targets. In the core of the proposed system there is a *Real Time Database* that is used for the necessary communication within the system.

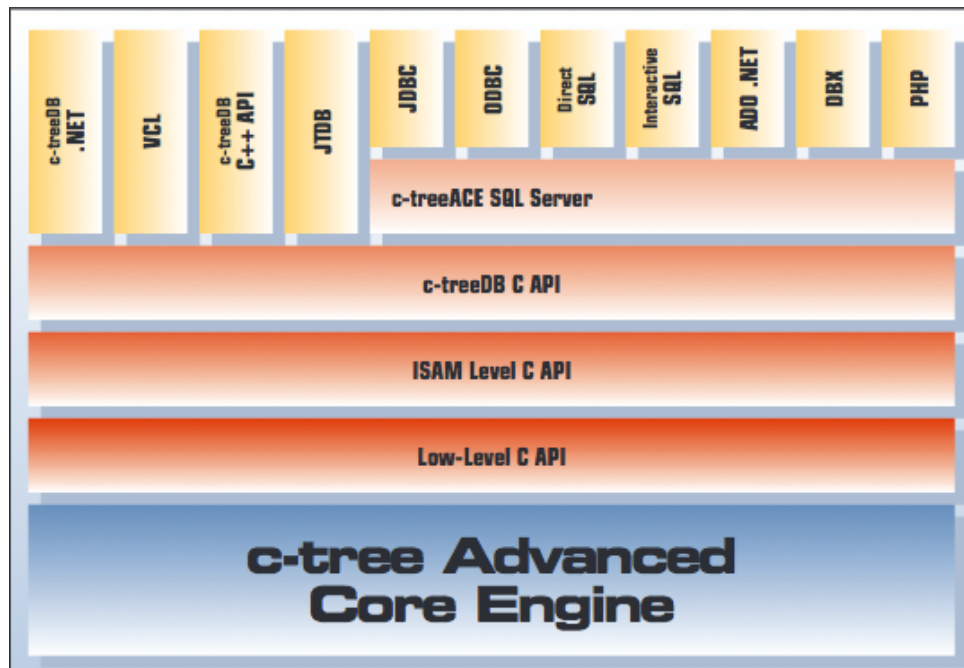
The project report is organized as follows. In Section C, the Technical Background is explained. Section D presents in full detail the approach of this project. The Results of the implemented algorithm are being discussed in Section E, and finally in Section F are the concluding remarks, the future work and the references. An Appendix is also included, where all code is provided.

## C. TECHNICAL BACKGROUND

The core of this project is basically divided into two basic components. The first vital part is the *Real Time Database* that is used to store the necessary information for the functionality of this project. It is also used for communication purposes between the different systems used during the experiments, as it will be discussed in the sections below. The second major component of this project is the algorithmic logic that allows the user to steer a population of simple robots to *exact* heading angles with a common input command.

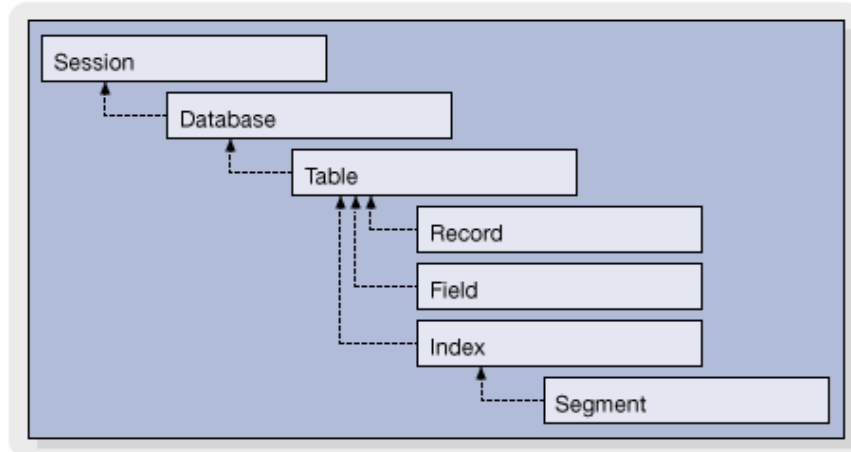
### i) *FairCom's Real Time Database*

*FairCom's* software, the *c-treeACE Database* [5], has been utilized to keep track of the necessary data during the experiments. This technology is based on a highly scalable, multi threaded compact core written in the *C* language that is the core engine of the *c-treeACE Server*.

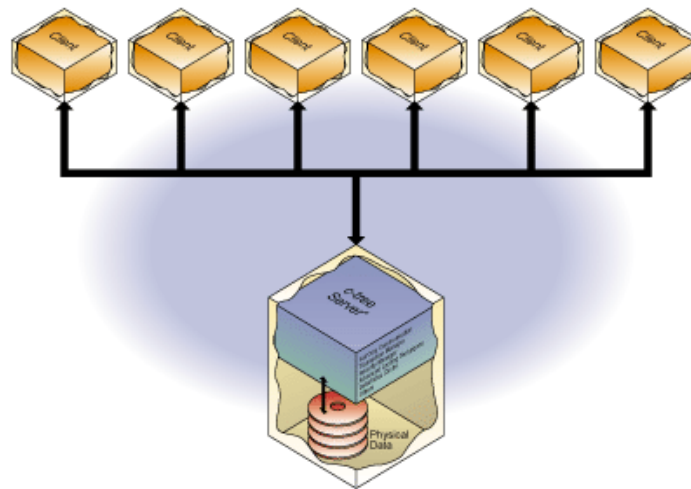


For the need of this project, the *c-treeDB C++ API* was used. The *c-treeDB C++ API* provides libraries that allow *C++* programs to access the *c-tree database core* via function calls. *c-treeDB* utilizes the simplified concepts of *sessions*, *databases*, and *tables* in addition to the standard concepts of *records*, *fields*, *indices* and *segments*.

**c-treeDB Relationships**



When designing the system, simplicity was the primary requirement. For this reason the single *c-treeACE* architectural model was adopted. In this model, one *c-tree Server* serves all clients. Because this model involves only one *c-tree Server*, there is no application routing logic and no synchronization of data among multiple servers. The application simply connects to the server and accesses the data.



The tradeoff for the simplicity of this approach is that the client load and scalability of the database system is limited to the capacity of the machine on which the *c-tree Server* runs. Also, the availability of the system is determined by the availability of the single *c-tree Server* process and the machine on which it runs. If a software or hardware component failure renders the *c-tree Server* process or its data inaccessible, the availability of the system is directly affected.

## ii) Robot Properties

The algorithm that is presented in the next section controls a collection of simple robots. The configuration of the  $i$ th robot is described by  $q_i = [x_i, y_i, \theta_i]^T$ . The global control inputs are the forward speed  $u \in \mathbb{R}$  and the turning rate  $\omega \in \mathbb{R}$ .

For the functionality of the algorithm, an assumption has to be made about the robot population. More specifically, each robot has a *unique non-zero* parameter  $\varepsilon_i$ :  $|\varepsilon_i| \neq |\varepsilon_j| \forall i, j$  that scales the turning rate. These  $\varepsilon_i$  values may rise from a stochastic process [2] or as design decisions. In the specific case of this project the values are uniformly distributed in  $[\frac{1}{2}, \frac{3}{2}]$ .

The *kinematics* of each robot are given by

$$\dot{q} = u(t) * \begin{bmatrix} \cos \theta_i \\ \sin \theta_i \\ 0 \end{bmatrix} + \varepsilon_i * \omega(t) * \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \quad (1)$$

## D. PROPOSED METHOD

Given  $q(0)$ ,  $q_{goal}$ , the control problem for regulating positions is to find  $u(t)$  and  $\omega(t)$  such that  $\lim_{t \rightarrow \infty} \left\| \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} - \begin{bmatrix} x_{goal} \\ y_{goal} \end{bmatrix} \right\| = 0, \forall q(0), q_{goal}$ . Previous work [6] has proven that the *heading angles*  $\theta_i$  are *not* controllable. This project presents the extension of this previous work by implementing an algorithm that controls the *relative* final heading – the *bearing* to a fixed target from each robot. To accomplish this, the algorithm exploits the fact that the final heading of the robots after applying an input sequence can be pre-computed.

This project's robotic system is modeled with a discrete-time model. During the first stage of step  $k$  the robots turn in place  $\varphi$ , while in the second stage they move in linear velocity  $u(k)$ . Under these constraints, (1) can be simplified in the following simple form:

$$\begin{bmatrix} x_i(k+1) \\ y_i(k+1) \end{bmatrix} = \begin{bmatrix} x_i(k) \\ y_i(k) \end{bmatrix} + \begin{bmatrix} \cos(\theta_i(0) + \varepsilon_i * k * \varphi) \\ \sin(\theta_i(0) + \varepsilon_i * k * \varphi) \end{bmatrix} * u(k) \quad (2)$$

So, after  $k$  moves under this motion model each robot's heading is  $\theta_i(k) = \theta_i(0) + \varepsilon_i * k * \varphi$ . If a distance  $d$  and a bearing angle  $a$  (counter clock-wise angle from the robot's heading to a vector toward the target) are given and the  $i$ th robot's target is at  $(x_{t,i}, y_{t,i})$ , the *desired final position* can be calculated as shown:

$$\begin{bmatrix} x_i(k) \\ y_i(k) \end{bmatrix} = \begin{bmatrix} x_{t,i} \\ y_{t,i} \end{bmatrix} + \begin{bmatrix} \cos(\theta_i(0) + \varepsilon_i * k * \varphi + a) \\ \sin(\theta_i(0) + \varepsilon_i * k * \varphi + a) \end{bmatrix} * d.$$

This control equation enables a variety of configurations, such as facing the goal position, forming a perimeter around it or looking out of it. The pseudo code of the algorithm that steers a robot population exactly to arbitrary *range*  $d$  and *bearing*  $a$  locations relative to targets in a *finite* number of steps is presented below:

```

 $i$ : # robots
 $k$ : # steps
 $p_{i,0} = (x_o, y_o, th_o)^T$ : starting state for  $i$ th robot
 $p_{i,F} = (x_F, y_F, th_F)^T$ : final state for  $i$ th robot
 $\varphi$ : turning rate
 $\varepsilon_i$ : unique non-zero parameter for  $i$ th robot
 $d_i$ : distance from target
 $\alpha_i$ : bearing to the target


---


 $\forall i$ :
    // Desired final position
     $p_{i,D} = p_{i,F} - \begin{bmatrix} \cos(th_{i,0} + \varepsilon_i * k * \varphi + \alpha_i) \\ \sin(th_{i,0} + \varepsilon_i * k * \varphi + \alpha_i) \end{bmatrix} * d_i$ 

    // Encode all heading info
     $\forall k$ :
         $B_i(k) = \begin{bmatrix} \cos(th_{i,0} + \varepsilon_i * k * \varphi) \\ \sin(th_{i,0} + \varepsilon_i * k * \varphi) \end{bmatrix}$ 
    end
end
 $\forall k$ :
    // All robots' headings for  $k$  step
     $B_k = [B_0(k); B_1(k); \dots; B_{n-1}(k)]$ 
end

    // Controllability Matrix
     $C_k = [B_0, B_1, \dots, B_{k-1}]$ 

    // Control Input Sequence
     $u_i[0, 1, \dots, k-1] = C_k^{-P} * (p_{i,D} - p_{i,0})$ 


---


 $\forall i$ :
     $\forall k$ :
         $p_{i,k+1} = p_{i,k} + B_i(k) * u_i(k)$ 
    end
end

```

To control each robot's position,  $C_k$  is required to be full rank. If the  $\varepsilon_i$  values are unique, there always exists a sequence of  $\varphi_k$  values that make  $C_k$  full rank and the robots converge exactly. If  $k$  is twice the number of robots, then  $C_k$  is almost always ill conditioned which leads to very large control commands. On the contrary, the best results are obtained when  $k$  is almost equal to four times the number of robots and the *non-zero* parameters  $\varepsilon_i$  are evenly distributed.

It is also important to mention two bounds concerning the turning rate  $\varphi$  and the number of steps  $k$ :

$$\varphi < \frac{\pi}{\max_{i \in [1,n]} \varepsilon_i} , \quad k \geq \frac{2\pi}{\min_{i \neq j \in [1,n]} |\varepsilon_i - \varepsilon_j|} .$$

The outcome of the above algorithm is a *sequence of  $k$  control inputs*  $u_{[0,\dots,k-1]}$  that brings the system to the desired final state. Each robot receives this control sequence and utilizes it along with the current position and the current heading information to determine its position at the next step.

## E. RESULTS

Based on this control algorithm, a system was built to test the efficiency of the implemented algorithm. The *MobileSim* robot simulator and the *Aria API* were used to create a population of robots and navigate them throughout the step-by-step positions derived from the control algorithm. Also the *Real Time Database* was utilized to create three matrices and store all the required information about the robots and their states. The three matrices are shown below:

<b><i>TCoord (#Robots x 3):</i></b>				
<b><i>ID</i></b>	<b><i>X</i></b>	<b><i>Y</i></b>	<b><i>Th</i></b>	
...	...	...	...	
...	...	...	...	
...	...	...	...	
...	...	...	...	

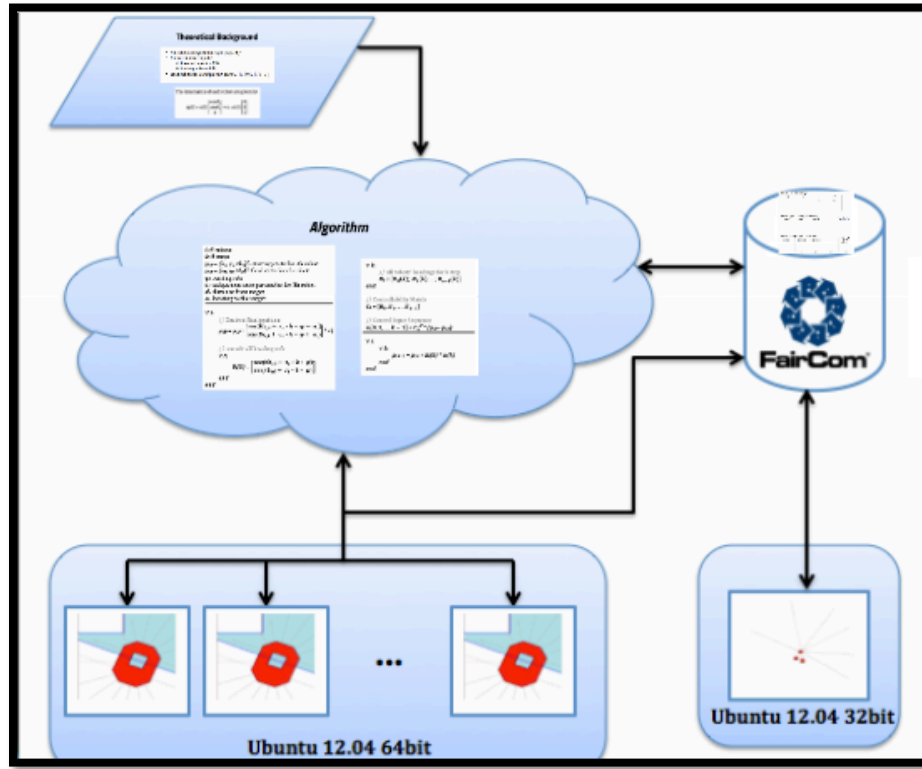
  

<b><i>TControl (2 * #Robots x #Steps)</i></b>				
<b><i>Step1</i></b>	<b><i>Step1</i></b>	<b><i>Step3</i></b>	<b><i>...</i></b>	<b><i>Step(k-1)</i></b>
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...

<b><i>TControlIn (#Steps x #Robots)</i></b>				
<b><i>Robot1</i></b>	<b><i>Robot2</i></b>	<b><i>Robot3</i></b>	<b><i>...</i></b>	<b><i>Robot (k-1)</i></b>
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...

Each robot updates its current position after every *completed* step of the algorithm in the first table. The second table contains all the heading information of the robots, while the last one is the actual control sequence that is derived from the algorithm and passed to each robot. The general diagram of the testing system is presented below:



The initial idea was to run each robot in a different machine and also have another main thread that actually visualizes the whole population in one single simulator window. Due to technical problems though, all robots run separately on their own simulator but under the same workstation. The algorithm calculations and results are being derived on a laptop and pushed in the Database for the robots to access and update.

In order to test the algorithm a variety of simulations took place. Short paths were chosen because under open-loop control, the true state diverges from the predicted one due to process noise (input commands, distance travelled, navigation errors, etc.). Each simulation starts with a group of  $n$  robots initialized randomly with  $x \in [0, 10000]$ ,  $y \in [0, 10000]$  and  $\theta \in [0, 180^\circ]$ . Also, the unique non-zero parameters  $\varepsilon_i \in \left[\frac{1}{2}, \frac{3}{2}\right]$ . The experiments investigated final robot position accuracy as a function of the number of steps  $k$ . The following pictures provide information for the final position of each robot after the completion of the simulation.

In the first shown experiment, a population of three robots was guided to position  $(5000, 5000)$  in a number of 19 steps, where  $d = 100$  and  $a = 0$ :



### 3 robots / 19 steps – Goal: (5000, 5000)

[illegible]

In the next experiment four robots approach the same goal position again in 19 steps, but this time the given distance to the target  $d = 50$  while bearing  $a$  remains the same:

### 4 robots / 19 steps – Goal: (5000, 5000)

**Terminal**

```

STARTING STATE: 10000.000000, 4712.661621, 84.827904
Step 0: nextX: 8511.134157, nextY: 4408.143073
Step 1: nextX: 8119.736573, nextY: 3603.546867
Step 2: nextX: 7823.400331, nextY: 4196.368130
Step 3: nextX: 6770.194576, nextY: 4399.959214
Step 4: nextX: 6827.340192, nextY: 4450.622948
Step 5: nextX: 6737.644236, nextY: 5714.524667
Step 6: nextX: 5828.835769, nextY: 6314.699494
Step 7: nextX: 5873.260477, nextY: 6330.047773
Step 8: nextX: 6053.342951, nextY: 6870.116483
Step 9: nextX: 6162.248309, nextY: 6709.017840
Step 10: nextX: 7364.863925, nextY: 6636.713657
Step 11: nextX: 8094.745007, nextY: 7478.104540
Step 12: nextX: 8092.373731, nextY: 7489.698737
Step 13: nextX: 8767.491271, nextY: 7161.287319
Step 14: nextX: 9138.393816, nextY: 7346.692198
Step 15: nextX: 9116.684721, nextY: 7234.388673
Step 16: nextX: 8762.522897, nextY: 7633.860797
Step 17: nextX: 6590.033844, nextY: 7479.684483
Step 18: nextX: 4983.670947, nextY: 5047.260171

POSITION REACHED ! ! !

```

**Terminal**

```

STARTING STATE: 0.000000, 830.316101, 14.945690
Step 0: nextX: -1488.865843, nextY: 525.797553
Step 1: nextX: -2230.123908, nextY: 24.695452
Step 2: nextX: -1864.893930, nextY: 577.740958
Step 3: nextX: -1661.302674, nextY: 1630.946743
Step 4: nextX: -1645.999401, nextY: 1556.125029
Step 5: nextX: -936.370006, nextY: 506.401783
Step 6: nextX: -27.561946, nextY: -93.773137
Step 7: nextX: -73.708756, nextY: -84.852674
Step 8: nextX: -631.463315, nextY: -198.930523
Step 9: nextX: -470.364377, nextY: -90.025112
Step 10: nextX: 193.561774, nextY: 915.317503
Step 11: nextX: 404.962671, nextY: 2008.923715
Step 12: nextX: 402.591299, nextY: 2020.517953
Step 13: nextX: 823.054092, nextY: 1398.546031
Step 14: nextX: 1169.070782, nextY: 1170.037313
Step 15: nextX: 1056.767123, nextY: 1191.746355
Step 16: nextX: 1579.801841, nextY: 1298.722759
Step 17: nextX: 3384.143787, nextY: 2518.486827
Step 18: nextX: 4990.510187, nextY: 4950.908810

POSITION REACHED ! ! !

```

It can be easily noticed that this time the robots actually came closer to the goal state than before, where the given distance was 100 instead of 50.

In the picture shown below, five robots are guided towards position  $(-5000, -5000)$  in 30 steps with a distance of 25 and a bearing angle of  $\pi$ :

### 5 robots / 30 steps – Goal: $(-5000, -5000)$

```
Terminal
Step 9: nextX: 9899.284596, nextY: -10297.556924
Step 10: nextX: 10372.189580, nextY: -10840.673399
Step 11: nextX: 10182.821152, nextY: -10904.208014
Step 12: nextX: 10184.798598, nextY: -10875.594548
Step 13: nextX: 9261.493589, nextY: -10416.195143
Step 14: nextX: 8266.121235, nextY: -11282.890647
Step 15: nextX: 8379.954408, nextY: -11622.179165
Step 16: nextX: 7677.928708, nextY: -11573.665836
Step 17: nextX: 7415.587675, nextY: -12100.921287
Step 18: nextX: 7188.531305, nextY: -11840.153657
Step 19: nextX: 6668.653135, nextY: -12014.575506
Step 20: nextX: 6710.328459, nextY: -11411.498058
Step 21: nextX: 5080.680006, nextY: -10600.650387
Step 22: nextX: 3952.699605, nextY: -11582.811400
Step 23: nextX: 3937.051974, nextY: -11536.172308
Step 24: nextX: 3213.461938, nextY: -11486.168791
Step 25: nextX: 3509.196674, nextY: -10891.798678
Step 26: nextX: 1870.697050, nextY: -9010.034189
Step 27: nextX: 123.663510, nextY: -9596.174267
Step 28: nextX: 258.683906, nextY: -7642.306037
Step 29: nextX: -5018.852415, nextY: -5016.416161

POSITION REACHED ! ! !
█

Step 29: nextX: -5001.723403, nextY: -5024.940521

POSITION REACHED ! ! !
█

Step 29: nextX: -4983.583471, nextY: -5018.853810

POSITION REACHED ! ! !
█

Step 29: nextX: -5011.136017, nextY: -5022.382823



POSITION REACHED ! ! !
█

Step 29: nextX: -4992.047953, nextY: -5023.701729

POSITION REACHED ! ! !
█
```

Finally, the last picture presents a population of ten robots trying to approach the goal position in 60 steps at a distance of 50 and with a bearing angle of  $\pi/2$ :

## 10 robots / 60 steps – Goal: (5000, 5000)

 Terminal	 Terminal
Step 39: nextX: 2418.563735, nextY: 5257.742884	Step 39: nextX: 6518.798579, nextY: 6445.599333
Step 40: nextX: 2646.329076, nextY: 5502.748033	Step 40: nextX: 6617.097165, nextY: 6125.846055
Step 41: nextX: 2594.093477, nextY: 4951.348660	Step 41: nextX: 6098.005642, nextY: 6319.014074
Step 42: nextX: 3178.695867, nextY: 4018.989783	Step 42: nextX: 5165.648761, nextY: 5734.409006
Step 43: nextX: 3875.669767, nextY: 3759.627036	Step 43: nextX: 5095.515400, nextY: 4994.055877
Step 44: nextX: 3950.751960, nextY: 3782.708581	Step 44: nextX: 5153.046096, nextY: 4940.573372
Step 45: nextX: 3973.065892, nextY: 3814.093685	Step 45: nextX: 5191.016897, nextY: 4946.987699
Step 46: nextX: 3950.621447, nextY: 4429.586037	Step 46: nextX: 5479.324665, nextY: 5491.242561
Step 47: nextX: 3407.474508, nextY: 5086.409343	Step 47: nextX: 5124.682790, nextY: 6266.261756
Step 48: nextX: 3247.495994, nextY: 5123.083553	Step 48: nextX: 4964.703892, nextY: 6302.935613
Step 49: nextX: 3948.759324, nextY: 5443.975385	Step 49: nextX: 5559.018453, nextY: 6794.394777
Step 50: nextX: 4348.640734, nextY: 6198.849312	Step 50: nextX: 5527.888339, nextY: 7648.075540
Step 51: nextX: 4331.762968, nextY: 6298.758055	Step 51: nextX: 5445.307593, nextY: 7706.787662
Step 52: nextX: 4508.160088, nextY: 6134.773220	Step 52: nextX: 5675.520751, nextY: 7777.559962
Step 53: nextX: 3864.510983, nextY: 6195.748145	Step 53: nextX: 5450.035404, nextY: 7171.623890
Step 54: nextX: 2286.394580, nextY: 5206.239857	Step 54: nextX: 6439.543835, nextY: 5593.507617
Step 55: nextX: 1654.983086, nextY: 3509.475633	Step 55: nextX: 8241.913871, nextY: 5422.769657
Step 56: nextX: 1794.054265, nextY: 3057.090448	Step 56: nextX: 8564.155122, nextY: 5769.401643
Step 57: nextX: 1652.421167, nextY: 3157.786666	Step 57: nextX: 8593.102490, nextY: 5598.049118
Step 58: nextX: 2889.004868, nextY: 3202.879740	Step 58: nextX: 7499.641902, nextY: 6177.287208
Step 59: nextX: 5048.736435, nextY: 4988.825961	Step 59: nextX: 4951.265848, nextY: 5011.172546
POSITION REACHED ! ! !	POSITION REACHED ! ! !
Step 59: nextX: 4951.264566, nextY: 5011.171986	Step 59: nextX: 5014.692257, nextY: 4952.204199
POSITION REACHED ! ! !	POSITION REACHED ! ! !
Step 59: nextX: 4985.308202, nextY: 5047.792743	Step 59: nextX: 4985.307356, nextY: 5047.792864
POSITION REACHED ! ! !	POSITION REACHED ! ! !
Step 59: nextX: 4965.955949, nextY: 4963.378961	Step 59: nextX: 5034.042610, nextY: 5036.619373
POSITION REACHED ! ! !	POSITION REACHED ! ! !
Step 59: nextX: 5034.044687, nextY: 5036.619889	Step 59: nextX: 5048.736876, nextY: 4988.826856
POSITION REACHED ! ! !	POSITION REACHED ! ! !

The results of these tests have shown that the paths increase in efficiency from  $k = 2n$  to  $k = 4n$ , but then decrease as the turning moves dominate. The results also shown that robot *collisions* are very likely to happen when the number of robots increases. More specifically, the probability of collision increases quadratically with the number of robots, thus providing a hard-limit on the open-loop control of the robot population.

## F. CONCLUSIONS / FUTURE WORK

This project examined controlling large populations of simple robots that each receives exactly the *same* input commands. A control algorithm was implemented to steer a robot population with unique turning rates to desired distance and bearing values in a finite number of steps. The results of the algorithm were validated in simulation.

There are several limitations to this project up until now. As far as the algorithm concerns, since it is an open-loop algorithm there is a realistic restriction in the number of steps. There is also a limitation in the number of robots since the control policy does not account for collisions.

Another problem was due to the robot navigation in the *MobileSim* environment. Although the intermediate positions of each robot are calculated correctly as shown in the Results Section, due to time limitations the navigation throughout these positions in the simulator is not optimal. Hence, the visualization of each robot's path is not accurate even though, again, then control algorithm derives the correct positions.

In the future versions of this project features as the following will be implemented:

- Optimize the *MobileSim* robot navigation control in order to achieve better accuracy in the robot simulations
- Implement the scenario where each robot runs in a different workstation and a main thread visualizes the population's movement
- Extend the implementation in order to support multiple goal positions to allow robots to form shapes, letters, etc.
- Improve Database access by using callback functions
- Others

## REFERENCES

- [1] S. Tottori, L. Zhang, F. Qiu, K. Krawczyk, A. Franco-Obregón, and B. J. Nelson, "Magnetic helical micromachines: Fabrication, controlled swimming, and cargo transport," *Advanced Materials*, vol. 24, no. 811, 2012.
- [2] B. Donald, C. Levey, and I. Paprotny, "Planar microassembly by parallel actuation of MEMS microrobots," *J. of MEMS*, vol. 17, no. 4, pp. 789–808, Aug. 2008.
- [3] Y. Shirai, A. J. Osgood, Y. Zhao, K. F. Kelly, and J. M. Tour, "Directional control in thermally driven single-molecule nanocars," *Nano Letters*, vol. 5, no. 11, pp. 2330–2334, Feb. 2005.
- [4] P.-T. Chiang, J. Mielke, J. Godoy, J. M. Guerrero, L. B. Alemany, C. J. Villagómez, A. Saywell, L. Grill, and J. M. Tour, "Toward a light-driven motorized nanocar: Synthesis and initial imaging of single molecules," *ACS Nano*, vol. 6, no. 1, pp. 592–597, Feb. 2011.
- [5] [http://www.faircom.com/pdf/c-treeACE\\_Technology\\_Overview.pdf](http://www.faircom.com/pdf/c-treeACE_Technology_Overview.pdf)
- [6] A. Becker and T. Bretl, "Approximate steering of a unicycle under bounded model perturbation using ensemble control," *IEEE Trans. Robot.*, vol. 28, no. 3, pp. 580–591, Jun. 2012.
- [7] A. Becker and J. McLurkin, "Exact Range and Bearing Control of Many Differential-Drive Robots with Uniform Control Inputs", *IROS 2013*, Nov. 2013
- [8] <http://www.faircom.com/doc/knowledgebase/index.htm#cover.htm>
- [9] <http://www.faircom.com/doc/ctreep/index.htm#cover.htm>

## APPENDIX

```

/*****
*                               MAIN CONTROL .CPP                               *
*****/

#define _USE_MATH_DEFINES
#define DEFAULT_PORT 8101

// Algorithm user inputs
#define ROBOT_NUM          4
#define STEPS_NUM          20
#define TURN_RATE          M_PI / 4
#define TARG_DIST          50
#define TARG_BEAR          0

// Preprocessor definitions and includes
#include "ctdbsdk.hpp" // c-tree headers

// Other includes
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <math.h>
#include <time.h>
#include "opencv2/core/core.hpp"

// Struct to pass arguments in thread routines
struct data {
    int id;
} pid[ROBOT_NUM];

// Global declarations
CTSession    MySession(CTSESSION_CTDB);
CTDatabase    MyDatabase(MySession);
CTTable      TCoord(MyDatabase), TControl(MyDatabase),
TControlIn(MyDatabase);

void *pid_init(void *arg);

// Function declarations
void Init(void);
void ManageTables(void);
void Update_Coords(CTTable *table, int i, double x, double y,
double theta);
void Display_Coords(CTTable *table);
void Save_Control(CTTable *table, int row, int col, double val);
void Display_Control(CTTable *table);
void Save_Inputs(CTTable *table, int row, int col, double val);
```



```

void Display_Inputs(CTable *table);
void Delete_Records(CTable *table);
void Cleanup(void);
void Check_Table_Mode(CTable *);
void Handle_Exception(CException);

int main (NINT argc, pTEXT argv[]) {
    int i, j;
    float e[ROBOT_NUM];
    double startX = 0.0, startY = 0.0, startTh = 0.0;
    pthread_t p_id[ROBOT_NUM];

    Init();

    ManageTables();

    //if session is active, retrieve the server name, user logon
    //                                     name and user password
    if (MySession.IsActive()) {
        printf("\nServer name      : %s\n",
              MySession.GetServerName().c_str());
    }

    //if database is active, retrieve the database name and table
    //                                     count
    if (MyDatabase.IsActive()) {
        printf("Database          : %s\n",
              MyDatabase.GetName().c_str());
        printf("Connected tables : %d\n",
              MyDatabase.GetTableCount());
    }

    /*****
    *                               ALGORITHM PREPARATIONS                               *
    *****/
    srand (time(NULL));

    // Starting state -> p_init: 2 * ROBOT_NUM x 1
    cv::Mat p_init(2 * ROBOT_NUM, 1, CV_64F);
    float partition = 10000 / ROBOT_NUM;
    for (i = 0, j = 1; i < 2 * ROBOT_NUM && j <= ROBOT_NUM;
         i = i + 2, j++) {
        // y = [0, 5000]
        float random = ((float) rand()) / (float) RAND_MAX;
        startX = i * partition;
        startY = 10000 * random;
        startTh = 180 * random;

        p_init.at<double>(i, 0) = startX;
        p_init.at<double>(i + 1, 0) = startY;
    }
}

```

```

    // Update database
    Update_Coords(&TCoord, j, startX, startY, startTh);
}

// Final state -> p_final: 2 * ROBOT_NUM x 1
cv::Mat p_final(2 * ROBOT_NUM, 1, CV_64F);
for (i = 0; i < 2 * ROBOT_NUM; i = i + 2) {
    p_final.at<double>(i, 0) = 5000;
    p_final.at<double>(i + 1, 0) = 5000;
}

// Unique parameters e[i] = [0.5, 1.5]
float part = 1.0 / (ROBOT_NUM - 1);
for (i = 0; i < ROBOT_NUM; i++) {
    e[i] = (i * part) + 0.5;
    //printf("e%d = %f\n", i, e[i]);
}
double min = ROBOT_NUM, max = 0;
for (i = 0; i < ROBOT_NUM; i++) {
    for (j = i + 1; j < ROBOT_NUM; j++) {
        if (fabs(e[i] - e[j]) < min) {
            min = fabs(e[i] - e[j]);
        }
    }
    if (e[i] > max) {
        max = e[i];
    }
}
if (STEPS_NUM < ((2 * M_PI) / min)) {
    printf("Error! Number of steps too small!\n");
    printf("Min steps are %d\n",
        (int) ceil(((2 * M_PI) / min)));
    Cleanup();
    exit(0);
}
if (TURN_RATE > (M_PI / max)) {
    printf("Error! Turn rate too big!\n");
    printf("max is %f\n", (M_PI / max));
    Cleanup();
    exit(0);
}

// Desired Final Position -> p_des: 2 * ROBOT_NUM x 1
cv::Mat p_des(2 * ROBOT_NUM, 1, CV_64F);
for (int j = 0, i = 0; i < 2 * ROBOT_NUM && j < ROBOT_NUM;
    i = i + 2, j++) {
    p_des.at<double>(i, 0) = p_final.at<double>(i, 0) -
        (cos(startTh + (e[j] * STEPS_NUM * TURN_RATE) +
            TARG_BEAR) * TARG_DIST);
}

```

```

        p_des.at<double>(i + 1, 0) = p_final.at<double>(i + 1, 0) -
            (sin(startTh + (e[j] * STEPS_NUM * TURN_RATE) +
                TARG_BEAR) * TARG_DIST);
    }

    // Controllability Matrix C: 2n x k
    cv::Mat C;
    C.create(2 * ROBOT_NUM, STEPS_NUM, CV_64F);
    for (int k = 0; k < STEPS_NUM; k++) {
        for (int j = 0, i = 0; j < 2 * ROBOT_NUM && i < ROBOT_NUM;
            j = j + 2, i++) {
            C.at<double>(j, k) = cos(startTh +
                (e[i] * k * TURN_RATE));
            C.at<double>(j + 1, k) = sin(startTh +
                (e[i] * k * TURN_RATE));
            // Save Controllability Matrix to Database
            Save_Control(&TControl, j + 1, k, C.at<double>(j, k));
            Save_Control(&TControl, j + 2, k,
                C.at<double>(j + 1, k));
        }
    }

    cv::Mat u(STEPS_NUM, 1, CV_64F),
        sub(2 * ROBOT_NUM, 1, CV_64F),
        C_inv(STEPS_NUM, 2 * ROBOT_NUM, CV_64F);
    cv::subtract(p_des, p_init, sub, cv::noArray(), -1);
    cv::invert(C, C_inv, cv::DECOMP_SVD);

    // Control Inputs u: k x 1
    u = C_inv * sub;
    for (i = 0; i < STEPS_NUM; i++) {
        // Save Control Inputs to Database
        Save_Inputs(&TControlIn, i + 1, 0, u.at<double>(i, 0));
    }
    /*****/

    // Create threads
    for (i = 0; i < ROBOT_NUM; i++) {
        pid[i].id = i;

        if ((pthread_create(&p_id[i], NULL, pid_init, &pid[i]))
            != 0) {
            printf("Error! Can't create thread!\n");
            Cleanup();
            exit(1);
        }
    }
}

```



```

// Wait for threads
for (i = 0; i < ROBOT_NUM; i++) {
    if (pthread_join(p_id[i], NULL)) {
        printf("Error! Thread can't join!\n");
        Cleanup();
        exit(1);
    }
}

printf("\nPress <ENTER> key to exit . . .\n");
getchar();

Cleanup();

return(0);
}

void Init(void) {
    CTBOOL create_session = NO, create_database = NO;
    printf("\nINITIALIZING..\n");

    try {
        ctdbStartDatabaseEngine();

        // connect to server
        MySession.Logon("FAIRCOMS", "ADMIN", "ADMIN");
        printf("Login to server DONE..\n");
    }
    catch (const CException &E) {
        if (E.GetErrorCode() == FNOP_ERR) {
            create_session = YES;
        }
        else {
            Handle_Exception(E);
        }
    }

    if (create_session) {
        try {
            // create new session
            MySession.Create("FAIRCOMS", "ADMIN", "ADMIN");
            printf("New Session created..\n");
            // connect to server
            MySession.Logon("FAIRCOMS", "ADMIN", "ADMIN");
            printf("Login to server DONE..\n");
        }
        catch (const CException &E) {
            Handle_Exception(E);
        }
    }
}

```

```

try {
    // connect to the database
    MyDatabase.Connect("MyDatabase");
    printf("Database connected..\n");
}
catch (const CException &E) {
    if (E.GetErrorCode() == INOT_ERR) {
        create_database = YES;
    }
    else {
        Handle_Exception(E);
    }
}

if (create_database) {
    try {
        // create a new database MyDatabase
        MySession.CreateDatabase("MyDatabase", "");
        printf("New Database created..\n");
        // connect to the database
        MyDatabase.Connect("MyDatabase");
        printf("Database connected..\n");
    }
    catch (const CException &E) {
        Handle_Exception(E);
    }
}

}

void ManageTables(void) {
    char str[32];
    CTBOOL create_table = NO;
    CRecord MyRecord1(TCoord);
    CRecord MyRecord2(TControl);
    CRecord MyRecord3(TControlIn);

    printf("\nMANAGING TABLES..\n");

    try {
        // add TCoord
        MyDatabase.AddTable("TCoord", "");
        printf("TCoord added..\n");
        MyRecord1.SetDefaultIndex(CTDB_ROWID_IDXNO);
        TCoord.Open("TCoord", CTOPEN_NORMAL);
        printf("TCoord opened..\n");
        Delete_Records(&TCoord);
    }
    catch(...) {
        create_table = YES;
    }
}

```

```

}

if (create_table) {
    // create TCoord
    try {
        TCoord.AddField("ID", CT_INTEGER, 1);
        TCoord.AddField("X", CT_DOUBLE, 8);
        TCoord.AddField("Y", CT_DOUBLE, 8);
        TCoord.AddField("Th", CT_DOUBLE, 8);
        TCoord.Create("TCoord", CTCREATE_NORMAL);
        printf("TCoord created..\n");
        MyRecord1.SetDefaultIndex(CTDB_ROWID_IDXN0);
        TCoord.Open("TCoord", CTOPEN_NORMAL);
        printf("TCoord opened..\n");
    }
    catch (const CException &E) {
        Handle_Exception(E);
    }
    create_table = N0;
}
else {
    Check_Table_Mode(&TCoord);
}

// initialize TCoord with data
try {
    for(int i = 0; i < ROBOT_NUM; i++) {
        MyRecord1.Clear();

        MyRecord1.SetFieldAsNumber(0, i + 1);
        MyRecord1.SetFieldAsFloat(1, -1);
        MyRecord1.SetFieldAsFloat(2, -1);
        MyRecord1.SetFieldAsFloat(3, -1);

        // add record
        MyRecord1.Write();
    }
}
catch(CException &E) {
    Handle_Exception(E);
}

try {
    // add TControl
    MyDatabase.AddTable("TControl", "");
    printf("TControl added..\n");
    MyRecord2.SetDefaultIndex(CTDB_ROWID_IDXN0);
    TControl.Open("TControl", CTOPEN_NORMAL);
    printf("TControl opened..\n");
    Delete_Records(&TControl);
}

```

```

}
catch(...) {
    create_table = YES;
}

if (create_table) {
    // create TControl
    try {
        for (int i = 0; i < STEPS_NUM; i++) {
            sprintf(str, "Field%d", i);
            TControl.AddField(str, CT_DOUBLE, 8);
        }
        TControl.Create("TControl", CT_CREATE_NORMAL);
        printf("TControl created..\n");
        MyRecord2.SetDefaultIndex(CTDB_ROWID_IDXN0);
        TControl.Open("TControl", CT_OPEN_NORMAL);
        printf("TControl opened..\n");
    }
    catch (const CException &E) {
        Handle_Exception(E);
    }
    create_table = NO;
}
else {
    Check_Table_Mode(&TControl);
}

// initialize TControl with data
try {
    for(int i = 0; i < 2 * ROBOT_NUM; i++) {
        MyRecord2.Clear();
        for (int j = 0; j < STEPS_NUM; j++) {
            MyRecord2.SetFieldAsFloat(j, -1);
        }
        // add record
        MyRecord2.Write();
    }
}
catch(CException &E) {
    Handle_Exception(E);
}

try {
    // add TControlIn
    MyDatabase.AddTable("TControlIn", "");
    printf("TControlIn added..\n");
    MyRecord3.SetDefaultIndex(CTDB_ROWID_IDXN0);
    TControlIn.Open("TControlIn", CT_OPEN_NORMAL);
    printf("TControlIn opened..\n");
    Delete_Records(&TControlIn);
}

```

```

    }
    catch(...) {
        create_table = YES;
    }

    if (create_table) {
        // create TControlIn
        try {
            for (int i = 0; i < ROBOT_NUM; i++) {
                sprintf(str, "Field%d", i);
                TControlIn.AddField(str, CT_DOUBLE, 8);
            }
            TControlIn.Create("TControlIn", CT_CREATE_NORMAL);
            printf("TControlIn created..\n");
            MyRecord3.SetDefaultIndex(CTDB_ROWID_IDXNO);
            TControlIn.Open("TControlIn", CT_OPEN_NORMAL);
            printf("TControlIn opened..\n");
        }
        catch (const CException &E) {
            Handle_Exception(E);
        }
        create_table = NO;
    }
    else {
        Check_Table_Mode(&TControlIn);
    }

    // initialize TControlIn with data
    try {
        for(int i = 0; i < STEPS_NUM; i++) {
            MyRecord3.Clear();
            for (int j = 0; j < ROBOT_NUM; j++) {
                MyRecord3.SetFieldAsFloat(j, -1);
            }
            // add record
            MyRecord3.Write();
        }
    }
    catch(CException &E) {
        Handle_Exception(E);
    }
}

void Update_Coords(CTTable *table, int i, double x, double y,
double theta) {
    try {
        CTRecord MyRecord(table);
        MyRecord.Clear();
        MyRecord.Lock(CTLOCK_WRITE_BLOCK);
    }
}

```

```

        //printf("\nData updating!\n");
        if (MyRecord.FindRowid(i, CT_FIND_EQ)) {
            MyRecord.SetFieldAsFloat(1, x);
            MyRecord.SetFieldAsFloat(2, y);
            MyRecord.SetFieldAsFloat(3, theta);

            MyRecord.Write();
            //printf("\nData updated!\n");
        }
        else {
            printf("\nNo Record found with id %d!\n", i);
        }
        MyRecord.Unlock();
    }
    catch(const CException &E) {
        Handle_Exception(E);
    }
}

void Display_Coords(CTable *table) {
    CTBOOL found;
    int id;
    double x, y, theta;

    try {
        CRecord MyRecord(table);
        // read first record
        found = MyRecord.First();
        while (found) {
            id = MyRecord.GetFieldAsShort(0);
            x = MyRecord.GetFieldAsFloat(1);
            y = MyRecord.GetFieldAsFloat(2);
            theta = MyRecord.GetFieldAsFloat(3);

            printf("ID: %d -> X:      %f, Y:      %f, Theta: %f\n",
                    id, x, y, theta);

            // read next record
            found = MyRecord.Next();
        }
    }
    catch(CException &E) {
        Handle_Exception(E);
    }
}

void Save_Control(CTable *table, int row, int col, double val) {
    try {
        CRecord MyRecord(table);
        MyRecord.Clear();
    }
}

```

```

MyRecord.Lock(CTLOCK_WRITE_BLOCK);

//printf("\nData updating!\n");
if (MyRecord.FindRowid(row, CT_FIND_EQ)) {
    MyRecord.SetFieldAsFloat(col, val);

    MyRecord.Write();
}
else {
    printf("\nNo Record found with in [%d, %d]!\n",
                                                row, col);
}
MyRecord.Unlock();
}
catch (const CTEException &E) {
    Handle_Exception(E);
}
}

void Display_Control(CTTable *table) {
    CTBOOL found;
    double val;

    try {
        CTRecord MyRecord(table);
        // read first record
        found = MyRecord.First();
        printf("Ck\n");
        while (found) {
            printf("Row ID %d: ", (int) MyRecord.GetRowid());
            for (int i = 0; i < STEPS_NUM; i++){
                val = MyRecord.GetFieldAsFloat(i);
                printf(" %f ", val);
            }
            // read next record
            found = MyRecord.Next();
            printf("\n");
        }
    }
    catch(CTException &E) {
        Handle_Exception(E);
    }
}

void Save_Inputs(CTTable *table, int row, int col, double val){
    try {
        CTRecord MyRecord(table);
        MyRecord.Clear();
        MyRecord.Lock(CTLOCK_WRITE_BLOCK);
    }
}

```

```

        //printf("\nData updating!\n");
        if (MyRecord.FindRowid(row, CTFIND_EQ)) {
            MyRecord.SetFieldAsFloat(col, val);

            MyRecord.Write();
        }
        else {
            printf("\nNo Record found with in [%d, %d]!\n",
                                                           row, col);
        }
        MyRecord.Unlock();
    }
    catch (const CException &E) {
        Handle_Exception(E);
    }
}

void Display_Inputs(CTable *table) {
    CTBOOL found;
    double val;

    try {
        CRecord MyRecord(table);
        // read first record
        found = MyRecord.First();
        printf("u\n");
        while (found) {
            printf("Row ID %d: ", (int) MyRecord.GetRowid());
            val = MyRecord.GetFieldAsFloat(0);
            printf("%f\n", val);

            // read next record
            found = MyRecord.Next();
        }
    }
    catch(CException &E) {
        Handle_Exception(E);
    }
}

void Delete_Records(CTable *table) {
    CTBOOL found;

    try {
        CRecord MyRecord(table);
        // read first record
        found = MyRecord.First();

```



```

        while (found) {
            // delete record
            MyRecord.Delete();
            // read next record
            found = MyRecord.Next();
        }
    }
    catch (CException &E) {
        Handle_Exception(E);
    }
}

void Cleanup(void) {
    printf("\nCLOSING DOWN..\n");
    try {
        TCoord.ResetAll();

        // close TCoord
        TCoord.Close();
        TControl.Close();
        TControlIn.Close();
        printf("Tables closed..\n");

        MyDatabase.DeleteTable("TCoord", "");
        MyDatabase.DeleteTable("TControl", "");
        MyDatabase.DeleteTable("TControlIn", "");
        printf("Tables deleted..\n");

        // disconnect MyDatabase
        MyDatabase.Disconnect();
        printf("Database Disconnected..\n");

        // delete MyDatabase
        MySession.DeleteDatabase("MyDatabase");
        printf("Database Deleted..\n");

        // Session logout
        MySession.Logout();
        printf("Logout DONE..\n");
    }
    catch (const CException &E) {
        Handle_Exception(E);
    }
}

void Check_Table_Mode(CTable *table) {
    try {
        // get table create mode
        CTCREATE_MODE mode = table->GetCreateMode();
    }
}

```

```

        // check if table is under transaction processing control
        if ((mode & CTCREATE_TRNLOG)) {
            // change file mode to disable transaction processing
            mode ^= CTCREATE_TRNLOG;
            table->UpdateCreateMode(mode);
        }
    }
    catch (const CException &E) {
        Handle_Exception(E);
    }
}

void Handle_Exception(CException err) {
    printf ("\nERROR: [%d] - %s\n", err.GetErrorCode(),
                                                    err.GetErrorMsg());
    printf("*** Execution aborted *** \nPress <ENTER> key to
                                                    exit...\n");

    getchar();
    Cleanup();

    exit(1);
}

void *pid_init(void *arg) {
    int ret;
    char str[128];
    int id;
    struct data *my_data;

    // Pass the arguments
    my_data = (struct data *) arg;
    id = my_data->id;

    sprintf(str, "gnome-terminal -x sh -c
        '/usr/local/MobileSim/MobileSim -p %d --nomap ' &",
        DEFAULT_PORT + id);

    ret = system(str);
    if (ret != 0) {
        printf("\nError! MobileSim didn't launch..!\n");
    }

    usleep(2000000);

    sprintf(str, "gnome-terminal -x sh -c
        '../MobileRob/Release/MobileRob -rrtp %d %d'",
        DEFAULT_PORT + id, DEFAULT_PORT + id);

    ret = system(str);
    if (ret != 0) {
        printf("\nError! Robot didn't launch..!\n");
    }
}

```

```

        pthread_exit(NULL);
        return 0;
    }

/*****
*                               MOBILE ROBOT THREAD                               *
*****/

#define _USE_MATH_DEFINES
#define DEFAULT_PORT 8101

#include <Aria.h>
// #include <cmath>
#include "ctdbsdk.hpp" // c-tree headers

// Global declarations
CTSession    MySession(CTSESSION_CTDB);
CTDatabase    MyDatabase(MySession);
CTTable      TCoord(MyDatabase), TControl(MyDatabase),
TControlIn(MyDatabase);

// Function declarations
void Init(void);
void ManageTables(void);
void Add_Records(int i, double x, double y, double theta);
double Get_Record(CTTable *table, int row, int col);
void Cleanup(void);
void Check_Table_Mode(CTTable *);
void Handle_Exception(CTException);

int main(int argc, char **argv) {
    int port, myID, steps;
    double startX=0, startY=0, startTh=0, nextX, nextY,
                                                b0, b1, u, a;

    Init();

    ManageTables();

    //if session is active, retrieve the server name, user logon
                                                name and user password
    if (MySession.IsActive()) {
        printf("\nServer name      : %s\n",
                MySession.GetServerName().c_str());
    }

    //if database is active, retrieve the database name and
                                                table count
    if (MyDatabase.IsActive()) {

```

```

    printf("Database          : %s\n",
           MyDatabase.GetName().c_str());
    printf("Connected tables : %d\n",
           MyDatabase.GetTableCount());
}

Aria::init();
ArArgumentParser argParser(&argc, argv);
argParser.loadDefaultArguments();
ArRobot robot;
ArRobotConnector robotConnector(&argParser, &robot);

if(!robotConnector.connectRobot()) {
    ArLog::log(ArLog::Terse, "Could not connect to the
                           robot.");

    if(argParser.checkHelpAndWarnUnparsed()) {
        // -help not given, just exit.
        Aria::logOptions();
        Aria::exit(1);
        return 1;
    }
}

// Trigger argument parsing
if (!Aria::parseArgs() ||
    !argParser.checkHelpAndWarnUnparsed()) {
    Aria::logOptions();
    Aria::exit(1);
    return 1;
}

ArKeyHandler keyHandler;
Aria::setKeyHandler(&keyHandler);
robot.attachKeyHandler(&keyHandler);

robot.runAsync(true);

// Find connected port
port = atoi(argParser.getArg(1));
myID = port - DEFAULT_PORT + 1;
printf("\n%d -> CONNECTED TO PORT: %d\n", myID, port);

// turn on the motors, turn off amigobot sounds
robot.enableMotors();
robot.comInt(ArCommands::SOUNDTOG, 0);

// position the robot in a specific place
startX = Get_Record(&TCoord, myID, 1);
startY = Get_Record(&TCoord, myID, 2);

```

```

startTh = Get_Record(&TCoord, myID, 3);

ArRobotPacket pkt;
pkt.setID(ArCommands::SIM_SET_POSE);
pkt.uByteToBuf(0); // argument type: ignored.
pkt.byte4ToBuf((ArTypes::Byte4)startX);
pkt.byte4ToBuf((ArTypes::Byte4)startY);
pkt.byte4ToBuf((ArTypes::Byte4)startTh);
pkt.finalizePacket();
robot.lock();
robot.getDeviceConnection()->writePacket(&pkt);
robot.unlock();

// to move the robot to the desired pose.
ArPose myInitialPose;
myInitialPose.setPose(startX, startY, startTh);
robot.moveTo(myInitialPose);

printf("\nSTARTING STATE:    %f, %f, %f\n",
        robot.getX(), robot.getY(), robot.getTh());

// Start approaching desired final state
steps = TControl.GetFieldCount();
float currX = startX;
float currY = startY;
float currTh = startTh;

for (int i = 0; i < steps; i++) {
    b0 = Get_Record(&TControl, ((2 * myID) - 1), i);
    b1 = Get_Record(&TControl, (2 * myID), i);
    u = Get_Record(&TControlIn, i + 1, 0);
    nextX = currX + b0 * u;
    nextY = currY + b1 * u;
    printf("Step %d: nextX: %f, nextY: %f\n", i, nextX, nextY);

    // Go to next position
    a = (atan2((nextY - currY), (nextX - currX)) * 180) / M_PI;

    if (a != 0) {
        robot.lock();
        robot.setRotVel(10);
        robot.unlock();
        while (fabs(a - robot.getTh()) > 2) {
            // Do nothing
            printf("Step %d ->a: %f, Th: %f, diff: %f\n",
                    i, a, robot.getTh(), fabs(a - robot.getTh()));
        }
    }
    robot.lock();
    robot.setRotVel(0);
}

```

```

robot.setVel(100);
robot.unlock();
if (a != 90 && a != -90) {
    while (fabs(nextX - robot.getX()) > 250) {
        // Do nothing
        printf("Step %d ->X: %f, Y: %f, diffX: %f,
            diffY: %f\n", i, robot.getX(),
            robot.getY(), fabs(nextX - robot.getX()),
            fabs(nextY - robot.getY()));
    }
}
else {
    while (fabs(nextY - robot.getY()) > 250) {
        // Do nothing
        printf("Step %d ->X: %f, Y: %f, diffX: %f,
            diffY: %f\n", i, robot.getX(), robot.getY(),
            fabs(nextX - robot.getX()),
            fabs(nextY - robot.getY()));
    }
}
robot.lock();
robot.setVel(0);
robot.unlock();

//currX = nextX;
//currY = nextY;
currX = robot.getX();
currY = robot.getY();
currTh = robot.getTh();

Add_Records(myID, currX, currY, currTh);
//printf("Step %d: actualX: %f, actualY: %f\n",
    i, robot.getX(), robot.getY());

}
printf("\nPOSITION REACHED ! ! !\n");

// wait for robot task loop to end before exiting the program
robot.waitForRunExit();

Aria::exit(0);

return 0;
}

void Init(void) {
    printf("\nINITIALIZING..\n");

    try {

```

```

        ctdbStartDatabaseEngine();

        // connect to server
        MySession.Logon("FAIRCOMS", "ADMIN", "ADMIN");
        printf("Login to server DONE..\n");
    }
    catch (const CException &E) {
        Handle_Exception(E);
    }

    try {
        // connect to the database
        MyDatabase.Connect("MyDatabase");
        printf("Database connected..\n");
    }
    catch (const CException &E) {
        Handle_Exception(E);
    }
}

void ManageTables(void) {
    printf("\nMANAGING TABLE..\n");
    CRecord MyRecord1(TCoord);
    CRecord MyRecord2(TControl);
    CRecord MyRecord3(TControlIn);

    // Open TCoord
    try {
        MyRecord1.SetDefaultIndex(CTDB_ROWID_IDXN0);
        TCoord.Open("TCoord", CTOPEN_NORMAL);
        printf("TCoord opened..\n");
    }
    catch(const CException &E) {
        Handle_Exception(E);
    }
    Check_Table_Mode(&TCoord);

    // TControl
    try {
        MyRecord2.SetDefaultIndex(CTDB_ROWID_IDXN0);
        TControl.Open("TControl", CTOPEN_NORMAL);
        printf("TControl opened..\n");
    }
    catch(const CException &E) {
        Handle_Exception(E);
    }
    Check_Table_Mode(&TControl);

    // Open TControlIn
    try {

```

```

        MyRecord3.SetDefaultIndex(CTDB_ROWID_IDXNO);
        TControlIn.Open("TControlIn", CTOPEN_NORMAL);
        printf("TControlIn opened..\n");
    }
    catch(const CException &E) {
        Handle_Exception(E);
    }
    Check_Table_Mode(&TControlIn);
}

void Add_Records(int i, double x, double y, double theta) {
    try {
        CRecord MyRecord(TCoord);
        MyRecord.Clear();
        MyRecord.Lock(CTLOCK_WRITE_BLOCK);

        if (MyRecord.FindRowid(i, CTFIND_EQ)) {
            MyRecord.SetFieldAsFloat(1, x);
            MyRecord.SetFieldAsFloat(2, y);
            MyRecord.SetFieldAsFloat(3, theta);

            MyRecord.Write();
        }
        MyRecord.Unlock();
    }
    catch(const CException &E) {
        Handle_Exception(E);
    }
}

double Get_Record(CTable *table, int row, int col) {
    double val;
    try {
        CRecord MyRecord(table);
        MyRecord.Clear();

        if (MyRecord.FindRowid(row, CTFIND_EQ)) {
            val = MyRecord.GetFieldAsFloat(col);
        }
        else {
            val = -1;
            printf("\nError! Element [%d, %d] missing!\n", row,
col);
        }
    }
    catch(const CException &E) {
        Handle_Exception(E);
    }

    return val;
}

```



```

}

void Cleanup(void) {
    printf("CLOSING DOWN..\n");
    try {
        TCoord.ResetAll();

        // close TCoord
        TCoord.Close();
        TControl.Close();
        TControlIn.Close();
        printf("Tables closed..\n");

        //MyDatabase.DeleteTable("TCoord", "");
        //printf("Table deleted..\n");

        // disconnect MyDatabase
        MyDatabase.Disconnect();
        printf("Database Disconnected..\n");

        // delete MyDatabase
        MySession.DeleteDatabase("MyDatabase");
        printf("Database Deleted..\n");

        // Session logout
        MySession.Logout();
        printf("Logout DONE..\n");
    }
    catch (const CTEException &E) {
        Handle_Exception(E);
    }
}

void Check_Table_Mode(CTTable *table) {
    try {
        // get table create mode
        CTCREATE_MODE mode = table->GetCreateMode();

        // check if table is under transaction processing control
        if ((mode & CTCREATE_TRNLOG)) {
            // change file mode to disable transaction processing
            mode ^= CTCREATE_TRNLOG;
            table->UpdateCreateMode(mode);
        }
    }
    catch (const CTEException &E) {
        Handle_Exception(E);
    }
}

```

```
void Handle_Exception(CException err) {  
    printf ("\nERROR: [%d] - %s\n", err.GetErrorCode(),  
err.GetErrorMsg());  
    printf("*** Execution aborted *** \nPress <ENTER> key to  
exit...\n");  
    getchar();  
    Cleanup();  
  
    exit(1);  
}
```

**YouTube video link:** <http://youtu.be/bq6A6ZnImfI>