Robotic's memories

Jose Antonio Gallardo, *G9*

***Abstract*—Robot's behaviour is a deep invesigation field, nowadays, robots can move, carry items and mimic nearly every physical activities that humans can perform, on this memories, the implementation of some robot methods is explained.**

***Index Terms*—Robots, movement, behaviour**

I. Introduction

THIS document is the controller component roadmap, all activities and expansions are explained here.

II. Activity 1 - Timer.h implementation

A. Activity introduction

First lines are include macros which allow us to use standard C functions, thread allows us to use posix threads, chrono is used to sleep for a concrete milliseconds amount of time, functional is used for binding callback function, future is used to retrieve thread variables result, and cstdio and iostream are used for input/output related tasks.

B. Timer.h private vars

The two private variables are go and period, go controls if the timeout thread should run the code or not, and period is the period that timeout sleeps before doing the task, variables are atomic so multithreaded execution can be safe.

C. Timer.h functions

Timer.h have 4 functions:

1) Connect: Receives a parameter which type is callback and contains the function that is invoked on every timeout event if timeout is started.
2) Start and stop functions: Manage private class variables using functions so variables are treated as atomic in order to control thread execution.
3) SetPeriod: Change the timeout period.

III. Activity 2 - Introduction to robot control

A. Activity introduction

In this activity, robot should be able to run across the map avoiding obstacles using laser data to detect them and some parameters to tune movement and rotation speed.

Activity goal is to cover the largest amount possible of map when running for 5 minutes. After some trials, the improvements below lead to get a efficient moving and to land a significatively amount of terrain.

B. Controller improvements

1) Intelligent turn control based on laser angle, so robot can choose the best turn side on every turn, laser data angle is sortened by -1.14 in the most left measurement and 1.14 in the rightest one, so using a flag in 0, we can tell the robot to choose between left and right turns, I did this one expecting the robot motion seems less dummy and more realistic, and is the result I got.
2) Acceleration system, based on a real driver, on larger speeds, sensors are activated between a sort period, in order to control the speed, speed starts at 300 mm/s, and on large straight paths, its up to 900, the speed is the denominator in a sleep sentence, so its sleeps less when speeds are higher, this one aim was to approach more map in the same time, so this was implemented in order to improve swept component results.
3) Trimmed laser data, to get a better obstacles info, this one is neccesary so the threshold can be better tunned and for the robot not to being dumb on the corners, this and a modification on simpleworld.xml is needed so the robot dont believe itself a obstacle.

C. Results

After five runs, following results were obtained:

Run	Percentage
1	64.8%
2	60.8%
3	61.44%
4	60.48%
5	61.28%
Avg	61.76%

The results above were measured by running the robot in a controlled enviroment per 5 minutes. The percentage shows the amount of land covered by the robot related with the total amount of land. The final result is the average of 5 differents execution.

IV. Activity 3 - Sending robot to a target location

A. Activity introduction

In this activity, robot should be able to run into a given location without caring about obstacles. In order to handle mouse clicks, a new component RCISMousePicker is needed.

Using this component and some algebra principles, component have to perform a plane transformation.

Target is a thread safe structure which prevents multiple threads accessing simultaneosly using a mutex and some c++ techniques such as multiple return functions(std::tuple).

Target coordinates are given in absolute reference system(Center of map is (0,0)) and needs to be transformed into robot coordinates(Robot is (0,0)), to do this, to equations were given to us.

B. Transformation

The first equation:

$$x' = R * y + t$$

Gives us x' coordinates performing a rotation(R) based on the robot angle(α) to the target location(y), and then, the resulting vector is added to the absolute robot coordinates(t), as we can see in the following image, this equation don't apply the desired transformation so is discarded.

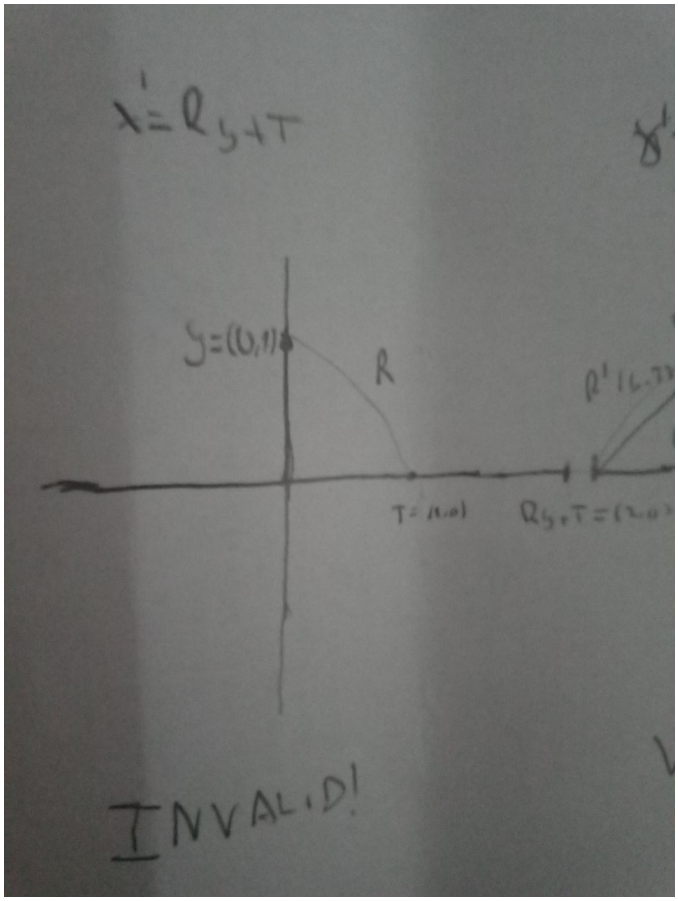


Fig. 1: Equation on a plane

The second equation:

$$x' = R' * (y - t)$$

Gives us x' coordinates multiplying the inverse of the robot angle(α) to the subtraction between target(y) coordinates and absolute robot coordinates(t), this transformation gives us the relative coordinates we are looking for.

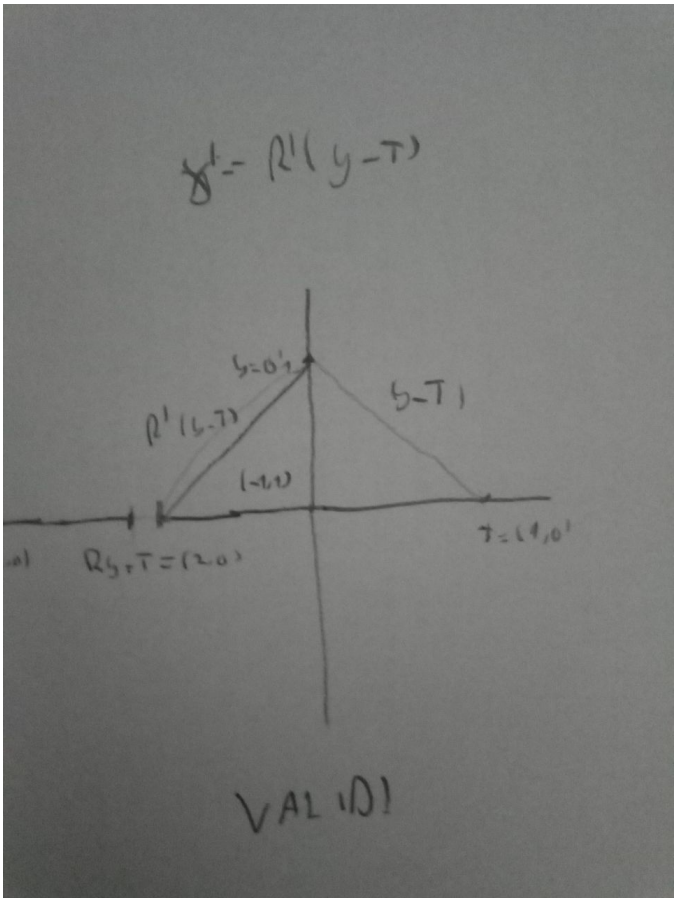


Fig. 2: Equation on a plane

C. Smooth movement

Rotation and movement speed needs to be tuned to make the movement smoother, in order to achieve this, two factors are calculated:

$$f1(x) = \begin{cases} 1, & \text{if } x \leq 200 \\ \frac{x}{200}, & \text{if } 0 \geq x \geq 200 \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Being x the norm of the relative target vector.

$$f2(\alpha) = gaussian(\alpha) \quad (2)$$

F2 is the gaussian with $\mu = 0$ and $\sigma = 0.2$ of the rotation angle.

Speed values are then adjusted as shown in equation below:

$$(speed, angle) = (MaxValue * F1 * F2, \alpha) \quad (3)$$

V. Activity 4 - Avoiding obstacles

A. Activity introduction

This activity expands Activity 3 component handling obstacle avoiding techniques with a state machine behaviour.

A state machine is implemented to avoid obstacles while going into the target, we have three main states: IDLE, on this state, robot is waiting for a new target. After a target is sent, GOTO state is triggered, when running into a target, laser is used as field-of-view and if there is an obstacle close to the robot, BUG state is triggered to pull out it using the BUG algorithm.

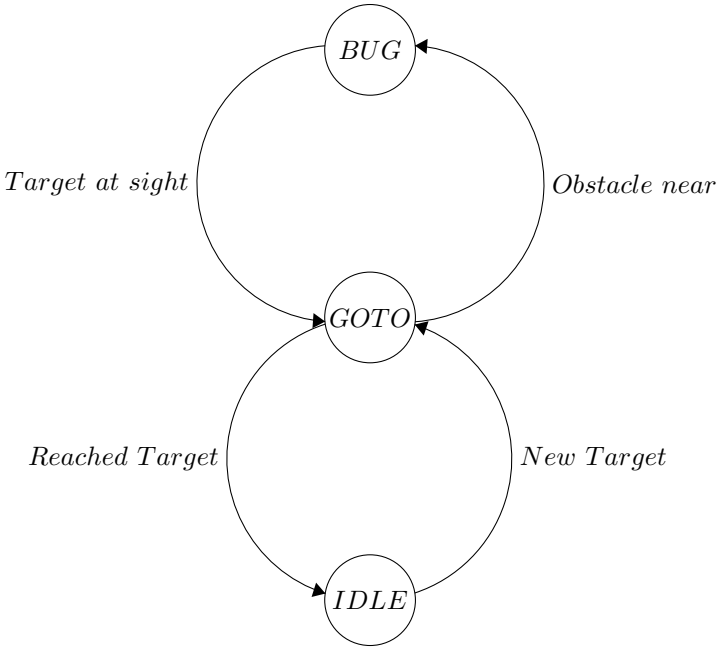


Fig. 3: State machine diagram

B. IDLE State

IDLE state manages the waiting period when robot is waiting for a new target. If a new target is picked with the mouse, GOTO state is called, otherwise, no task is performed.

C. GOTO State

GOTO state is the main state while robot is moving, first, GOTO state evaluates if there is a obstacle near, using a trimmed portion of the laser as explained in activity 2, if a obstacle is detected, BUG state is triggered.

If there is no obstacles near, InnerModel is used to transform target coordinates to the robot plane, then, distance between robot and target is checked and when reached, state is set to IDLE, if not reached, robot moves towards the target.

D. BUG State

Bug state got a pre-state, called GIRO, when BUG state is triggered, GIRO is launched, GIRO just turn the robot to the left until there is no obstacles near, then, BUG performs the following actions:

1) Checks if the target location is on robot field of view, or if he crossed the mline, if it is, trigger to GOTO.
2) If another obstacle is on view, calls GIRO again.
3) The state logic computes the movement and rotation speed and moves the robot fastened to the near walls until one of the above conditions are satisfied.

E. BUG Movement constants

In order to perform a smooth movement, rotation speed is fixed between (-0.5, 0.5) using and computed using a sigmoid function which main parameter is the distance between the wall we are following and the robot, and then, movement speed is calculated using the rotation speed as parameter.

VI. Activity 5 - Using Djikstra to avoid obstacles

A. Activity introduction

This activity consist on finding the optimal route using djikstra algorithm, djikstra algorithm take into account the cost from every possible route to the objective and return the optimal one.

B. Grid

To do this, we got a Grid class, our first step was to save a grid where boxes and walls were defined so algorithm could know those paths are forbbiden, to do this, a run per the entire map is performed so the Grid is updated.

The component auto-load the updated grid in the init, so user only need to click on the target to perform the pathfinding.

C. Pathfinding

When the user clicks a new target, algorithm stores the best route in a list of points, robot needs to pass over all of this points in order to reach the target without error.

To do this, Djikstra algorithm performs as follows:

1) Djikstra creates two vectors, one to store the minimal distance of each point and other to store the previous path points.
2) Djikstra checks every active points, and then proceed diferent if checking the target or checking any other point on the map.

3.1) If it's not on target, check every single neighbour and if $mindist(neighbour) > mindist(point) + cost(neighbour)$, update the active point to check to *neighbour* and store on the previous vector the checked *point*.

3.2) If it's on target, gets the optimal path by returning a ordened list of the points stored on the previous vector.

D. Curve smoothing

Curve is smoothed by implementing Bezier algorithm of grade *n*, being *n*, data is double interpolated based in the *n-1* curve based on linear combination of Berstein polynomials following Eq. 4

$$\mathbf{B}(t) = \sum_{i=0}^{n} \binom{n}{1} (1-t)^{n-1} t^i \mathbf{P}^i \tag{4}$$

where:

- t is the number of points conforming bezier curve (Approach accuracy)
- n is the grade of the curve, in this implementation, matchs Djikstra path computed points.
- $\binom{n}{1}(1-t)^{n-1}t^i\mathbf{P}^i$ is the Berstein polynomial of grade *n*, computed using a curve point ($\mathbf{P}^i$), the grade and the accuracy.

E. Component map drawing

Component also draws the map into a QT Canvas, map is drawn as a blank square where obstacles are marked in dark red color. Every tick, map updates robot position (Robot is drawn as a square with a rounded nose pointing robot's head) and if a point is picked, both, the Djikstra picked path and the Bezier curve are drawn. Bezier curve is drawn as a green line meanwhile Djikstra path is drawn in red.

F. Robot movement

After robot gets the points list, next step is to pass over all list points, to do this, a iterative process is performed, if there is a point, this point is the current target, after this point is reached, next point in the list is current target, until we reach the real target. this is done by modifying rotation and advance speed by using the same equations from Activity 3 (Eqs. 1, 2 and 3) with tweaked values for this problems.

VII. Activity 6 - Mission planning using April Tags and Djikstra

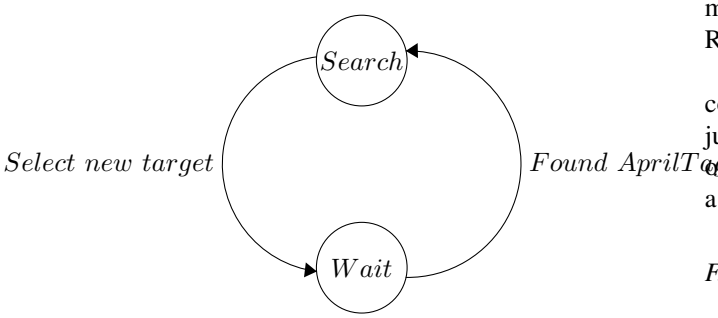
A. Activity introduction

This activity uses various components simultaneously to achieve the goal of making the robot visit some presetted location (Marked in map as April Tags). Components mentioned above are the Supervisor, who manages where AprilTags are found and sends movements commands to another component, the Movement Controller, this component sends the robot to the target location by picking a path as done in Activity 5, using Djikstra. When robot is alligned with target, Supervisor awaits the visual contact with AprilTag before sending robot to the next location, visual confirmation is provided by another component called Camera Controller.

B. Supervisor

As said before, supervisor manages the entire process by sending robot to a list of locations and by awaiting visual contact with the AprilTags. The list of locations is formed by the coordinates of all the AprilTags on the map, when an AprilTag is picked, is requeued in the end of the list, so robot can be looking out over the map until components are closed.

Supervisor works as a finite state machine with two states as shown in the figure below.



Those two states perform the following operations:

- **Search State**: While on search, supervisor pops the first position of the list, extract the target coordinates and then, push the position at the end of list. Then, target location is sent to Movement Controller and status is set as Wait.
- **Wait State**: Waits for robot to reach the target location, when reached, starts turning the robot until Camera Controller found the AprilTag, then, robot is sent to the next location by changing the state to Search.

C. Movement Controller

Movement controller is fully based on Activity 5 controlling model (Djikstra + Grid pathing), but implements a new interface that allows Supervisor to send the robot to different new targets.

This new interface implements methods to stop the robot, to go to a target or to turn the robot as a desired rotation speed.

Also, some modifications have been done to the grid so allows the new map, like the dimensions or the tile size. Grid's graphical mode have become optional so low-tier computers can run the model without problems, because when loading a large map, if graphical mode is set, computer starts to freeze.

D. Camera Controller

This controller uses RGBD proxy in order to find AprilTags on the robot field of view. When an AprilTag is found, the frame is stored in memory and using AprilTags API functions and QVec, we can find the relative position of the robot, and the ID of the found tag.

Found tags are sent to supervisor by publishing in the AprilTags component. This component is entirely based on the given AprilTagsComp, but have an added feature explained below.

This features in Telegram integration and allows the component to send a Telegram Message when an AprilTag is found. This process is made by using a Telegram bot that publishes a message in its feed when an AprilTag is seen. This could be useful by adding the possibility to detect intruders. When an intruder is seen, the patrol robots send to the security responsible phone the position of the intruder.

E. Launching the system

As the system is provided as a collection of components, running them 1 by 1 could be a tedious process, so a launching script have been built. This script works by pipelining commands to Yakuake console by openning new tabs that host the RCIS, RCNODE and the three components.

System can be vectorized by passing the folder of the three components for each robot, so, if you have two robots, you just need to pick up the components with changing port in one of the copies and launch the script passing the new folder as a parameter.

F. Testing environment

The new provided informatica-tercio map have been used as the test environment for the Mission Planner, after adding 4 new AprilTags, one at each corner of the map, the robot is able to visit them all in an ordered way.

VIII. Conclussions

Developing a system with a lot of components and activities should be always an iterative process starting from a scratch component and building the system from the little to the big as done here.

The entire source code of the activities above can be found at: https://github.com/jgallardst/practicasRobotica, source code will be make public after last activity deadline day.