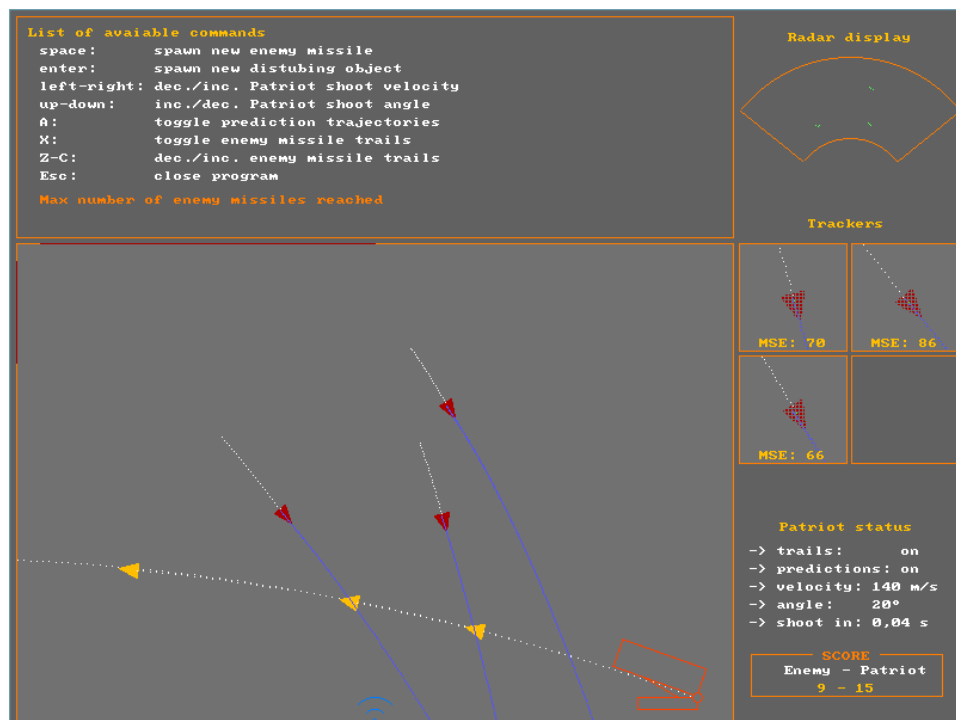


Patriot Project

Real Time Systems A.A. 2016/17
prof. Giorgio Buttazzo

Michael Mugnai
n°556448
m.mugnai@ymail.com



Delivery date: August 22, 2017

Contents

1	Assignment	3
2	Problem definition	3
3	Realization concepts	4
3.1	Missiles	4
3.1.1	Missiles characterization	4
3.2	Radar	5
3.3	Trackers	5
3.4	Parameters evaluation	6
3.4.1	Horizontal velocity	6
3.4.2	Vertical velocity	7
3.5	Rocket launcher	8
3.6	Missiles interception	8
4	Design choices	9
4.1	Graphics	9
4.2	Task and resources	10
5	Project files	11
6	Results	12

1 Assignment

Here's what the chosen assignment said:

Simulate a set of Patriot defense missiles that identify enemy targets, predict their trajectories and are launched to catch them.

Therefore the program has to simulate a two-dimensional piece of sky, monitor the presence of flying objects that would collide with the ground and intercept them.

Enemy missiles are supposed to have ballistic trajectory.

2 Problem definition

Assuming a two-dimensional space, the ground is represented as the baseline of the so-called *world box*. Let's suppose that the right side is a *safe zone*: no missile can come from here. In this way, the Patriot rocket launcher can be placed on the bottom right of our world box.

The problem can be subdivided in this parts:

1. a radar that monitors the sky and individuate every flying object;
2. a tracking camera that follows every object pointed out by the radar;
3. acquired object's positions is used to estimate velocities and accelerations, supposing ballistic trajectories;
4. if predicted positions correspond to real positions (within certain inaccuracy), the flying object has effectively a ballistic trajectory, and can be assumed as hostile;
5. rocket launcher evaluates estimated interception time with launcher's trajectory (fixed), then shoot at the right time (given by Patriot time-to-fly).

In this way, we can build up the program following a modular approach.

3 Realization concepts

What will follow summarizes the concepts behind each element of the program.

3.1 Missiles

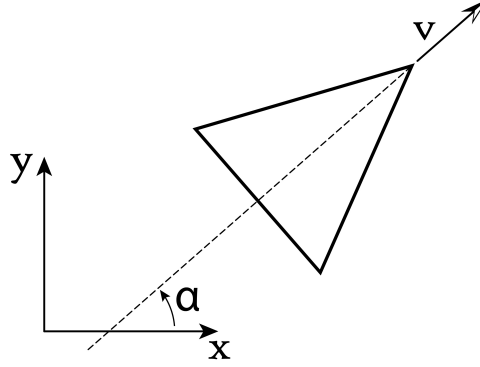


Figure 1: Missile geometry.

Missiles are central object of this program. They are characterized by ballistic motion: the only force applied is $m\mathbf{g} = [0 \ 0 \ -mg]^T$. Missiles can be categorized as *enemy missiles* or *Patriot missiles*.

In any case, the equations that govern missile's motion are:

$$\begin{cases} x = v_x t + x_0 \\ y = -\frac{1}{2}gt^2 + v_y t + y_0 \end{cases} \quad (1)$$

Where a more practical, step-increment formula can be defined:

$$\begin{cases} v_y^{(k+1)} = v_y^{(k)} - g \, dt_k \\ x_{k+1} = v_x \, dt_k + x_k \\ y_{k+1} = -\frac{1}{2}g \, dt_k^2 + v_y^{(k+1)} t + y_k \end{cases} \quad (2)$$

Notice that v_x is constant for the entire life of a missile.

Current missile orientation α (respect to horizon) can be recovered easily:

$$\frac{v_y}{v_x} = \frac{v \sin \alpha}{v \cos \alpha} = \tan \alpha \rightarrow \alpha = \arctan \frac{v_y}{v_x} \quad (3)$$

3.1.1 Missiles characterization

Initial values will determinate the “faction” of any missile:

1. **Enemy missiles:** these objects can spawn from *left side* or from *top side* of the world box, each one with equal probability. Initial position, orientation and velocity are generated randomly in a range that guarantees a collision with the ground inside the world box;
2. **Patriot missiles:** objects generated by the rocket launcher, thus with initial position and orientation fixed, meant to intercept enemy missiles.

3.2 Radar

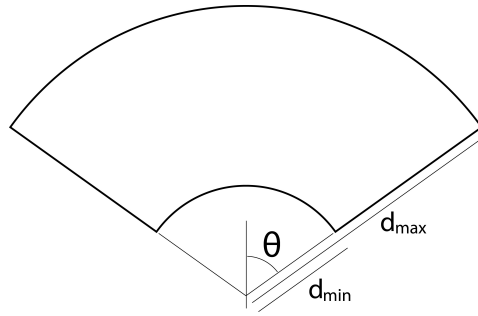


Figure 2: Radar geometry.

The radar is a device located ahead of the rocket launcher, in our world box is in the *mid point* of the ground. From radar's origin, a beam plumbs the sky radially, until a *max distance* is reached.

The purpose of the radar is to detect presence of flying objects, and communicate it to the trackers. When an object is already tracked, there's no benefit in scanning it anymore, therefore points too near to radar (thus to the ground) are useless to monitor: a *minimum distance* from radar's origin can be defined, and the beam starts only from there.

The plumbed area is, since the radar is centered on our world, specular on the vertical axis: given θ the angle with the horizon, the radar aperture is $\pi - 2\theta$.

What is seen by the radar is reproposed on *top-right corner* of the program.

3.3 Trackers

When the presence of something in the sky is detected, a tracking camera is responsible to establish what it is.

Focusing on a square of pixels of fixed dimensions, a moving object can be observed by letting the center of the camera chases the *barycenter* of what is currently seen.

The idea is simple: in every time-instant, pixels different from sky's color are taken into account in the evaluation of the center point; in particular, given a

square image with n pixels (x_i, y_i) different from sky color, the *centroid* vector position is:

$$\begin{bmatrix} c_x \\ c_y \end{bmatrix} = \frac{1}{n} \sum_{i=1}^n \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad (4)$$

Positions of subsequent time-instants are stored, allowing to estimate object's trajectory.

3.4 Parameters evaluation

Characterizing a ballistic trajectory leads to identify two main parameters: *horizontal velocity*, constant but unknown, and *vertical velocity*, that increase at every time-instant with a slope of g .

Since we can presume that tracker's detected positions are afflicted by error, what we're looking for is a trajectory that *minimize* its distance from any given point, instead of trying to evaluate an exact interpolation of them.

3.4.1 Horizontal velocity

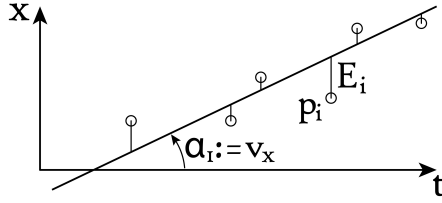


Figure 3: v_x evaluation.

As previously said, there's no acceleration along x 's axis, thus a (x, t) plot will exhibit a line, with inclination v_x . What we do now is to find the equation of the rect $f(t) = \alpha_0 + \alpha_1 t$ that has the minimum distance with each collected point. This leads to look for α_0 and α_1 that minimize $S_{(\alpha_0, \alpha_1)} := \sum_{i=1}^n |x_i - f(t_i)|^2$.

$$\min_{\alpha_0, \alpha_1} S_{(\alpha_0, \alpha_1)} \rightarrow \begin{cases} \frac{\partial S}{\partial \alpha_0} = -2 \sum_{i=1}^n (x_i - \alpha_0 - \alpha_1 t_i) = 0 \\ \frac{\partial S}{\partial \alpha_1} = -2 \sum_{i=1}^n (x_i - \alpha_0 - \alpha_1 t_i) t_i = 0 \end{cases} \quad (5)$$

$$\begin{cases} n\alpha_0 + \alpha_1 \sum_{i=1}^n t_i = \sum_{i=1}^n x_i \\ \alpha_0 \sum_{i=1}^n t_i + \alpha_1 \sum_{i=1}^n t_i^2 = \sum_{i=1}^n x_i t_i \end{cases} \quad (6)$$

$$\alpha_0 = \frac{\sum_{i=1}^n t_i^2 \sum_{i=1}^n x_i - \sum_{i=1}^n t_i \sum_{i=1}^n x_i t_i}{n \sum_{i=1}^n t_i^2 - (\sum_{i=1}^n t_i)^2} \quad (7)$$

$$\alpha_1 = \frac{n \sum_{i=1}^n t_i x_i - \sum_{i=1}^n t_i \sum_{i=1}^n x_i}{n \sum_{i=1}^n t_i^2 - (\sum_{i=1}^n t_i)^2} \quad (8)$$

The purpose of all of this is to estimate v_x , i.e. rect's inclination α_1 , thus for us the equation (8) is the only one relevant.

3.4.2 Vertical velocity

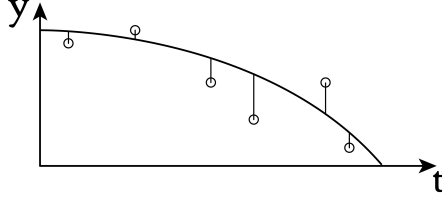


Figure 4: v_y evaluation.

In a similar way we can proceed to estimate v_y . Since vertical acceleration is constant (equals to $-g$), the (y, t) plot is parabolic, therefore $f(t) = \alpha_0 + \alpha_1 t + \alpha_2 t^2$, where $\alpha_2 = -g$.

Equation (6) changes as follows ($\frac{\partial S}{\partial \alpha_2} = 0$ is omitted because redundant):

$$\frac{\partial S}{\partial \alpha_0} = 0 \rightarrow n\alpha_0 + \alpha_1 \sum_{i=1}^n t_i = \overbrace{\sum_{i=1}^n x_i}^p - \alpha_2 \sum_{i=1}^n t_i^2 \quad (9)$$

$$\frac{\partial S}{\partial \alpha_1} = 0 \rightarrow \alpha_0 \sum_{i=1}^n t_i + \alpha_1 \sum_{i=1}^n t_i^2 = \underbrace{\sum_{i=1}^n x_i t_i}_q - \alpha_2 \sum_{i=1}^n t_i^3 \quad (10)$$

$$\alpha_0 = \frac{\sum_{i=1}^n t_i^2 p \sum_{i=1}^n x_i q}{n \sum_{i=1}^n t_i^2 - (\sum_{i=1}^n t_i)^2} \quad (11)$$

$$\alpha_1 = \frac{nq - \sum_{i=1}^n t_i p}{n \sum_{i=1}^n t_i^2 - (\sum_{i=1}^n t_i)^2} \quad (12)$$

As before, we're looking for α_1 , so only equation (12) is needed.

3.5 Rocket launcher

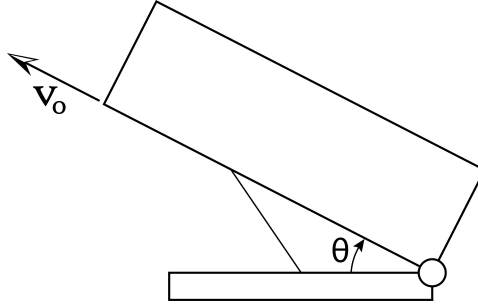


Figure 5: Rocket launcher geometry.

Patriot's launcher is setted in bottom left corner of world box, giving its back to the safe zone. Its orientation, combined with Patriot initial velocity, determines the path that every launched missile will follow. These parameters can be changed by user (pressing arrows), no need to say that they must produce paths that intersect enemy trajectories, otherwise there's no point of interception.

3.6 Missiles interception

The procedure used to estimate interception points between missiles' and Patriot's trajectories is widely described on the attached file "Missile_trajectories.pdf".

Summarizing briefly, *Cartesian equations* are extracted from time-dependent trajectory equations in order to identify *interception coordinates*; time is reintroduced and enemy and Patriot *time-to-fly* (from actual position to interception coordinates) is evaluated; a *wait time* is identified, as the difference between enemy's and Patriot's TTF. As long as wait time is positive, the rocket launcher waits, and when it reach zero, it shoot.

4 Design choices

Now let's talk about the choices made in the development of the program, both graphic and structural.

4.1 Graphics

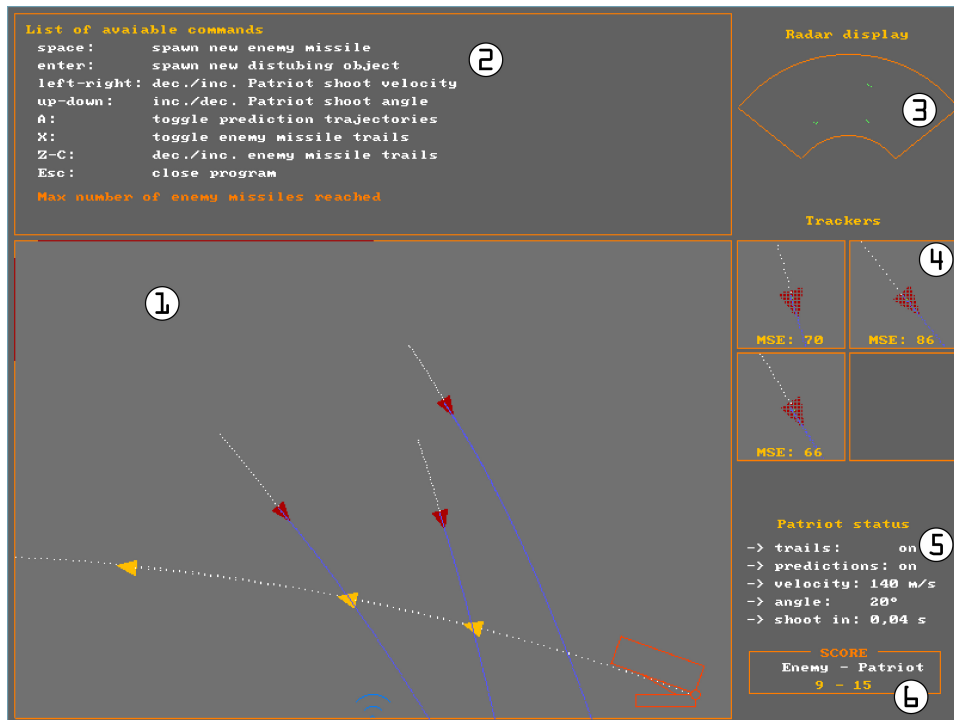


Figure 6: User interface.

The window (800×600 px) is subdivided in many parts:

1. the largest box is *world box*, containing the simulation of Patriot's rocket launcher, and the sky where enemy missile will fall;
2. on top-left there's a *quick brief* of available commands;
3. on top-right there's the *radar display*, where what is seen by the radar is reproduced;
4. below that, there are 4 *tracker displays*, where each tracker view is replicated, magnified by a scale factor (currently $\times 3$), and its trajectory MSE (Mean Square Error) is reported;
5. on center-right there are Patriot current parameters;

6. on bottom-right there's a *scoring board*.

Available commands are:

- *space bar*: spawn new enemy missile;
- *enter*: spawn new disturbing object;
- *left-right*: decrement/increment Patriot shoot velocity;
- *up-down*: increment/decrement Patriot shoot angle;
- *A*: toggle prediction trajectories;
- *S*: switch between tracker display parameters;
- *X*: toggle enemy missile trails;
- *Z-C*: decrement/increment enemy missile trails;
- *Esc*: close program.

Task logs like *deadline misses* and *utilization percentages* are printed in cosole win-
dow.

4.2 Task and resources

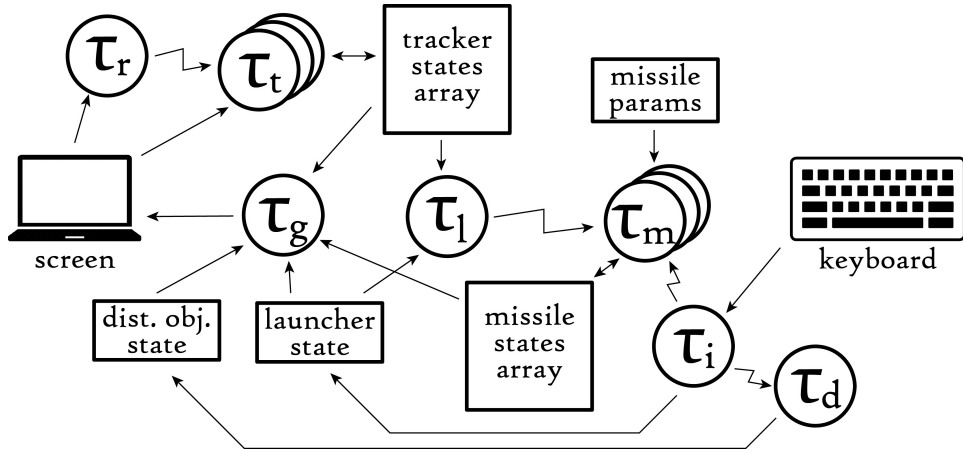


Figure 7: Tasks and resources scheme.

The program contain six different types of task, each one periodic, whose relations are expressed in Figure 7. Here's a detailed list, ordered by task priority:

1. *missile task* τ_m : it could be enemy's or Patriot's (depending on its initialization), it follows the physics laws imposed by ballistic trajectory; it has period of 20 ms;

2. *disturbing object task* τ_d : a flying object that has not ballistic trajectory; it has period of 20 ms;
3. *graphic task* τ_g : handles all graphic related functions; a double buffer is implemented in order to minimize screen flickering; it has period of 20 ms (50 fps);
4. *radar task* τ_r : scan a portion of world box in search of objects; it has period of 2 ms (in each iteration perform only a scan on one line), it can starts many tracker tasks;
5. *tracker task* τ_t : a camera that follows the object suggested by the radar and tries to estimate its trajectory; it has period of 20 ms;
6. *launcher task* τ_l : using the collected data from trackers, it tries to intercept enemy missiles, procucing its own missiles; it has period of 50 ms;
7. *interpeter task* τ_i : keyboard listener, it can change launcher state variables and start new enemy missiles; it has period of 40 ms.

Each task has its relative deadline equals to its period. The program is able to detect deadline misses and average utilization percentages (assumed as the sum of response times over its period).

Since multiple instances of missile and tracker tasks are necessities, a max value for each one is defined, and the task parameters array is structured as follows in Figure 8.

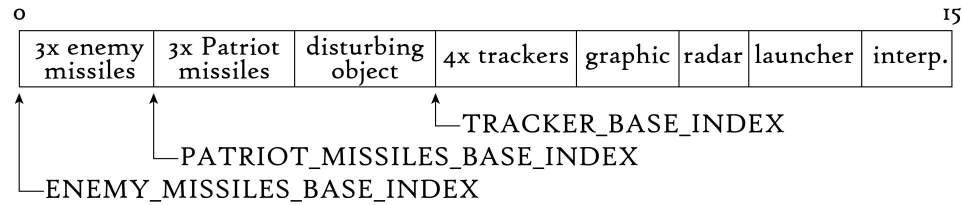


Figure 8: Task parameters array implementation.

5 Project files

Here's a description of the files contained in the project:

- *baseUtils.c/h*: general popruse utility functions, aimed to handle time, base task commands and Allegro library;
- *colors.h*: color definitions for every object; can be choosen between DARK_MODE and LIGHT_MODE;

- *common.c/h*: functions and tasks related to graphics and keyboard controls;
- *main.c*:
- *makefile*: gcc compiler instructions; available commands are *clean* (to clean all *.o files), *run* (to compile and run the program) and *rundebug* (to compile and start gdb debugger);
- *missiles.c/h*: functions and task regarding enemy's and Patriot's missiles behavior;
- *radar.c/h*: functions and task regarding radar behavior;
- *rocket_launcher.c/h*: functions and task regarding rocket launcher behavior and missile interception evaluation;
- *task_constants.c/h*: periods, relative deadlines and priorities for all tasks;
- *trackers.c/h*: functions and task regarding trackers behavior and trajectories evaluations.

6 Results

Even a Mean Square approximation on the sampled points seems unable to produce trajectory parameters sufficiently accurate to destroy enemy target with a single Patriot shoot. In fact, the oscillation of the error, presumably caused by a too low resolution of the world box (600×400 px, each pixel is 1 meter), makes the Patriot shoot up to a maximum of three times per target (once imposed a minimum delay between shoot of 300 ms).

Furthermore, objects with rectilinear trajectories (no accelerations) seems to be interpreted as ballistic trajectories where the parabola is almost collapsed in a rect. However this type of objects doesn't cause accidental shoots, due to their supposed collision time always positive.

Nevertheless, Patriot is capable of intercept and destroy enemy targets in many situations, once the objects are tracked for a suitable amount of time (usually at least one second). From that, a further improvement could be made by improving the radar, in order to achieve earlier target acquisitions.

List of Figures

1	Missile geometry.	4
2	Radar geometry.	5
3	v_x evaluation.	6
4	v_y evaluation.	7
5	Rocket launcher geometry.	8
6	User interface.	9
7	Tasks and resources scheme.	10
8	Task parameters array implementation.	11