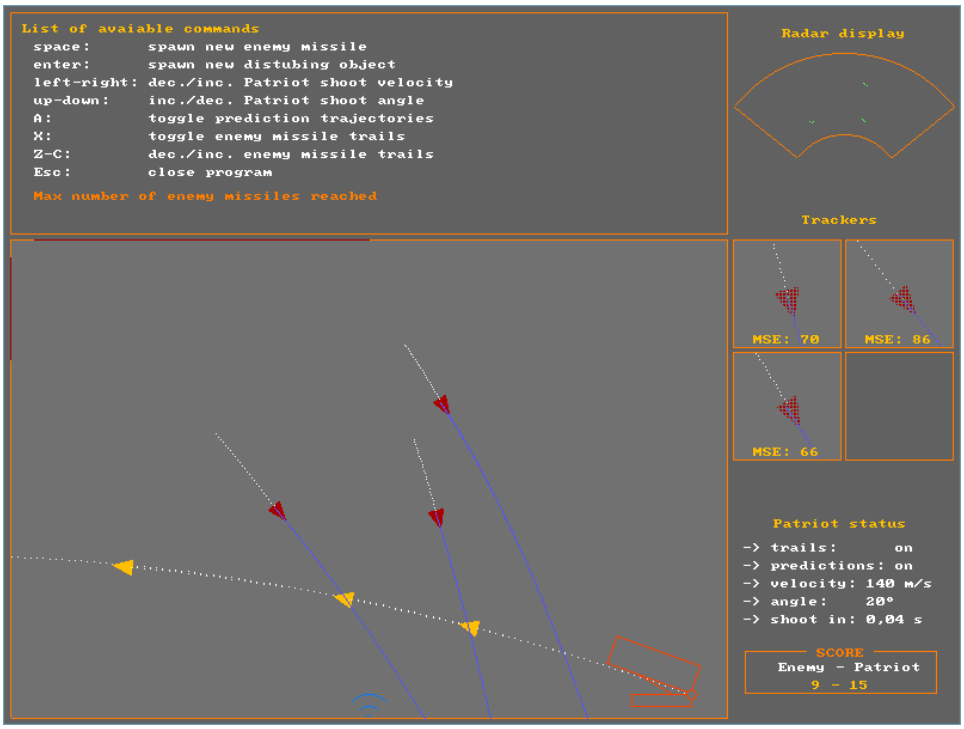
Patriot Project

Real Time Systems A.A. 2016/17

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



Delivery date: not today

Abstract

The delivery date. The report must include: the shared data structures, the tasks involved, a taskresource diagram, a discussion on the task parameters (how they were defined), and a set of experimental results. Task code must not be included in the report.

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 7
3.6 Missiles interception 8

4 Design choices **9**
4.1 Graphics 9
4.2 Task and resources 10

5 Results **10**

1 Assignment

Here's what the choosen 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 tarjectory.

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 effectivelly a ballistic trajectory, and can be assumed as obstile;
5. rocket laucher 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

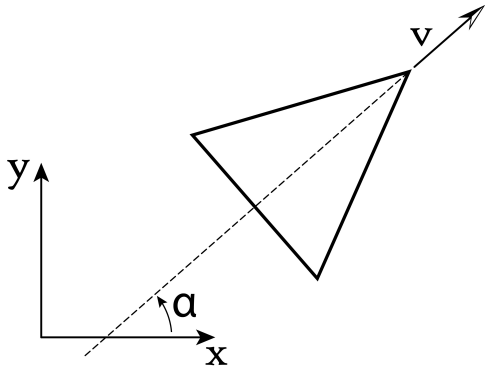


Figure 1: Missile geometry.

Missiles are central object of this program. They are caracterized by ballistic motion: the only force applied is $m\boldsymbol{g} = \begin{bmatrix} 0 & 0 & -mg \end{bmatrix}^T$. Missiles can be categorized as *enemy missiles* or *Patriot missiles*.

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

$$\begin{cases} x = v_x t + x_0 \\ y = -\frac{1}{2} g t^2 + v_y t + y_0 \end{cases} \tag{1}$$

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

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

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

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

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

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

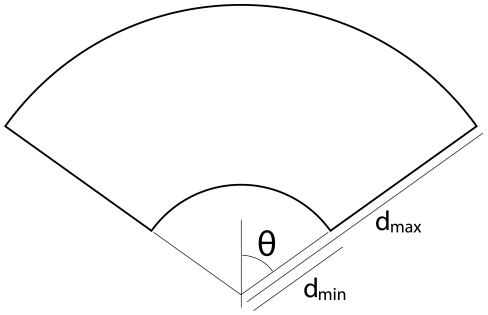


Figure 2: Radar geometry.

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 responsable to enstablish 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:

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

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

3.4 Parameters evaluation

Caracterizing 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 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

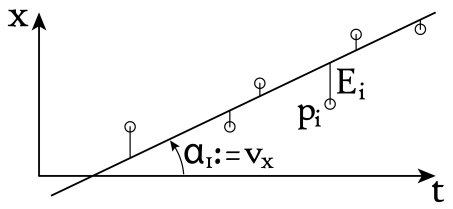


Figure 3: v_x evaluation.

As previously said, there's no acceleration along x's axis, thus a (x, t) plot will exibit 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 $S_{(\alpha_0,\alpha_1)} := \sum_{i=1}^{n} |x_i - f_{(t_i)}|^2$.

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

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

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

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

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

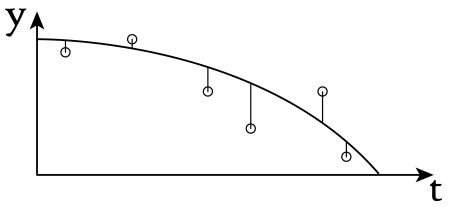


Figure 4: v_y evaluation.

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):

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

$$\frac{\partial S}{\partial \alpha_1} = 0 \rightarrow \alpha_0 \sum_{i=1}^{n} t_i + \alpha_1 \sum_{i=1}^{n} t_i^2 = \underbrace{\sum_{i=1}^{n} x_i t_i - \alpha_2 \sum_{i=1}^{n} t_i^3}_{q} \tag{10}$$

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

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

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

3.5 Rocket launcher

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.

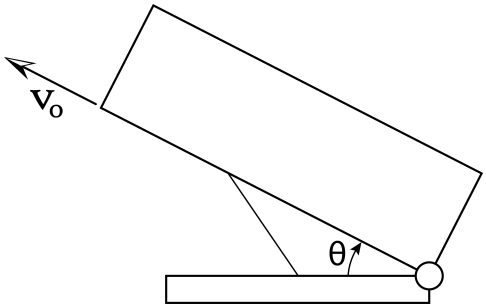


Figure 5: Rocket launcher geometry.

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

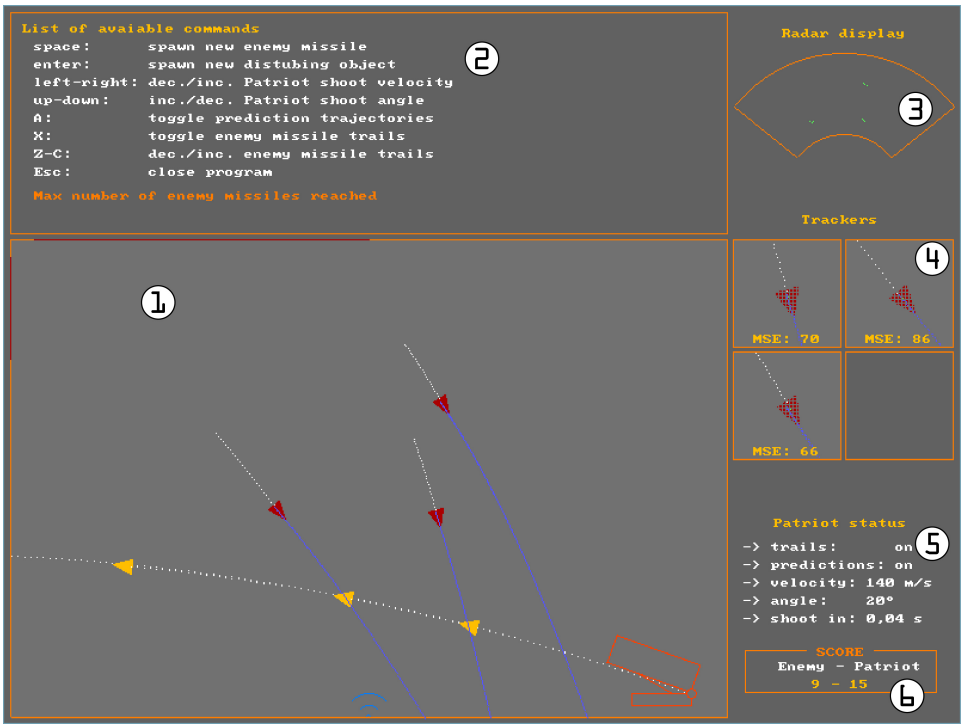


Figure 6: User interface.

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

1. the largest box is *world box*, cointaining 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 avaiable 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) si reported;
5. on center-right there are Patriot current parameters;

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

Avaiable commands are:

- *space bar*: spawn new enemy missile;
- *enter*: spawn new distubing object;
- *left-right*: decrement/increment Patriot shoot velocity;
- *up-down*: increment/decrement Patriot shoot angle;
- *A*: toggle prediction trajectories;
- *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 window.

4.2 Task and resources

5 Results

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