

# CANNON PROJECTILE GAME 1

---

ELECTRICAL ENGINEERING 20N  
Department of Electrical Engineering and Computer Sciences  
University of California, Berkeley

SIMON HONG, HSIN-I LIU, JONATHAN KOTKER, AND BABAK AYAZIFAR

---

## 1 Introduction

In this mini-project, we will bring together all of the ideas from lab 01, lab 02 and lab 03 to complete the framework for a game application, allowing us further practice in understanding and using the LabVIEW constructs that will be the most useful to us over the semester.

### 1.1 Mini-Project Goals

---

- Solidify understanding of LabVIEW constructs.
- Examine the usage of LabVIEW constructs and tools in a practical situation.

### 1.2 Checkoff Points

---

- 2. The Problem .....
- 3. The Framework ..... (50%)
- 4. Submission Rules .....
- 5. Submission Instructions .....
- 6. Acknowledgments .....

## 2 The Problem

---

This game will require the user to control parameters of a projectile cannon to hit a randomized target. The user will have control over the following inputs:

1. Angle: a value in degrees between 0 and 90
2. Power
3. Height: a value in metres
4. Launch
5. Stop

The front panel of the application will have the following display outputs:

1. Target Distance: a value in metres between 0 and 5000
2. Wind: a value in miles per hour (mph) between 0 and 100
3. Number of Attempts
4. Distance
5. Score
6. Projectile Graph

When the game begins, the `Score` and `Number of Attempts` will be initialized to 0 and a new number will be generated for `Target Distance` and `Wind` within the specified ranges. The user will then proceed to change the three parameters of the canon: `Angle (deg)`, `Power`, and `Height (m)`. Once the parameters are finalized, the user will press `Launch` and the projectile curve will *slowly* draw itself on the projectile graph based off the following equations:

$$x = \frac{330}{k} \times \text{Power} \times \left(1 - e^{\left(-\frac{kt}{100}\right)}\right) \cdot \cos(\theta), \text{ and}$$

$$y = \text{Height} + \left(3.3 \times \text{Power} \times \sin(\theta)t - 4.9t^2\right),$$

where  $k$  is a drag constant determined by the wind speed as per the following relationship:

$$k = \frac{\text{Wind}}{20}.$$

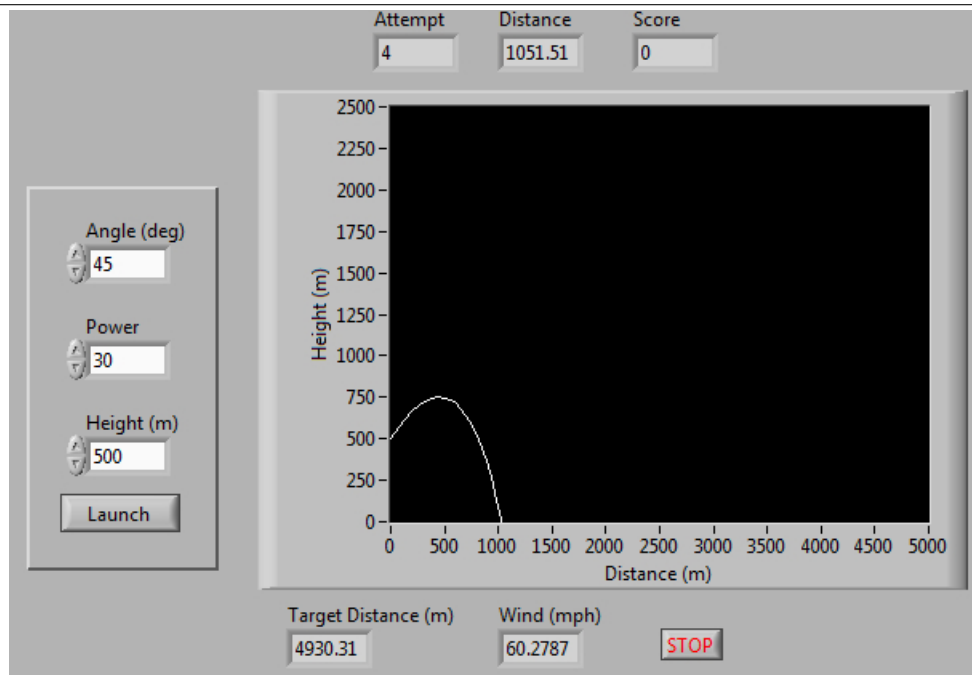
Notice that the  $\theta$  in the equations above need to be in *radians*, and not *degrees*.

If the user misses the target, the program will increment `Number of Attempts`, hold the same `Target Distance` and `Wind` values and wait for the user to attempt another launch. The `Distance` indicator will display the horizontal distance actually traveled by the projectile. However, if the user hits the target within 100 metres, a dialog box will pop up with the message `Direct Hit!` and the `Score` display will update based off the following formula:

$$\text{Score} = \text{Previous Score} + 1000 - 100 \times \text{Attempts}.$$

Once the score has been updated, a new set of `Target Distance` and `Wind` values are generated and the user can play another round. The front panel of the completed VI is shown in [Figure 1](#).


**Figure 1** Cannon Projectile Game Front Panel.



### 3 The Framework


Download and open `Cannon Projectile Game Framework.vi`. Navigate to the block diagram. Take a moment to explore the various blocks in the framework, and to reason out their purpose in the general framework.

1. The entire block diagram is enclosed in a large `While Loop`, which runs with a delay of 100 milliseconds to prevent unnecessary CPU slowdown and memory usage.
2. The leftmost `Case Structure` is used to generate the random values for the `Target Distance (m)` and `Wind (mph)` indicators on the front panel. **The `True` case is left incomplete.**
3. The `While Loop` inside the other, large `Case Structure` is used to generate the arrays of  $X$  and  $Y$  values, as per the formulae given in [section 2](#), using the parameters provided. The arrays are generated until the value of  $Y$  drops below zero, which is the `Loop Condition` of the `While Loop`. This is because, after  $Y$  drops below zero, the projectile is below the field of vision, and the points therefore plotted are irrelevant. **The generation of the  $X$  and  $Y$  values is left incomplete.**

The `For Loop` plots the projectile. Notice that it uses a `Bundle` () to collect two arrays into one **cluster**, and feeds the result to an `XY Graph`.

A **cluster** is merely a collection of items of different data types, like the `struct` data type in C. Since the cluster is fed to an `XY Graph`, the `XY Graph` ungroups the cluster, and considers the first object of the collection (here, an array) as the values for its  $X$ -axis. It then considers the second object (here, also an array) as the values for its  $Y$ -axis.

4. The mini-project specifications require us to plot the graph *slowly*. We will accomplish this in the following manner:

- (a) The `For Loop` will have a `Wait (ms)` block so that there will be a small delay between each iteration.
- (b) In each iteration  $i$ , we will only plot the values of the  $X$  and the  $Y$  arrays from 0 to  $i$ , to give the illusion of progressive plotting. The `Array Subset` blocks  already present in the framework will help.

**The slow plotting of the projectile in the XY Graph on the front panel needs to be implemented.**

5. The last, and rightmost, `Case Structure` determines what happens to the values of `Score` and `Attempts` depending on whether the user hits or misses the target. **The updating of the values of `Score` and `Attempts`, in both cases of the `Case Structure` is left incomplete. Also, there is no dialog box displaying `Direct Hit!` in the case of a successful hit.**

---

## 4 Submission Rules

1. Complete the framework as specified in [section 3](#), by addressing the issues in boldface. Your final VI must function as specified in [section 2](#).
2. Late submissions will *not* be accepted, except under unusual circumstances.
3. These exercises are recommended to be done *in groups of two*. Only one person need submit the required files, however.
4. You may *not* work in groups across lab sections.
5. The TA solution VI is available online as `Cannon Projectile Game Final.vi` to compare your answer against. Please do not check the block diagram of the solution: it is password-protected. We know that a few of you are 1337 hack3rz and can crack our password, but we think you will be better off (and safer!) in the long run if you complete the VI on your own effort.

---

## 5 Submission Instructions

1. Log on to [bSpace](#) and click on the `Assignments` tab.
2. Locate the assignment for `Mini-Project 1`.
3. Attach the final VI to the assignment (as a VI called `Cannon Projectile Game`), and a file called `PARTNERS.txt` containing the names of the students in the group.
4. This assignment is due **March 4, 2011** at 6 PM for all sections. Please do not wait until (literally) the last minute to submit your work though; [bSpace](#) stops allowing submissions precisely on the minute, and since it takes a while to upload and submit your work, you may not be able to complete your submission.

---

## 6 Acknowledgments

Special thanks go out to the teaching assistants (TAs) of the Spring 2009 semester (Vinay Raj Hampapur, Miklos Christine, Sarah Wodin-Schwartz), of the Fall 2009 semester (David Carlton, Judy Hoffman, Mark Landry, Howard Lei, Feng Pan, Changho Suh), and of the Spring 2010 semester (Xuan Fan, Brian Lambson, Kelvin So) for providing suggestions, ideas, and fixes to this guide.