Software Requirements Specification for Projectile

Samuel J. Crawford, Brooks MacLachlan, and W. Spencer Smith

November 3, 2023

Contents

1 Reference Material

This section records information for easy reference.

1.1 Table of Units

The unit system used throughout is SI (Système International d'Unités). In addition to the basic units, several derived units are also used. For each unit, the Table of Units lists the symbol, a description, and the SI name.

Table 1: Table of Units

Symbol	Description	SI Name
m	length	metre
rad	angle	radian
s	time	second

1.2 Table of Symbols

The symbols used in this document are summarized in the Table of Symbols along with their units. Throughout the document, symbols in bold will represent vectors, and scalars otherwise. The symbols are listed in alphabetical order. For vector quantities, the units shown are for each component of the vector.

Table 2: Table of Symbols

Symbol	Description	Units
a	Scalar acceleration	$\frac{\text{m}}{\text{s}^2}$
a^c	Constant acceleration	$\frac{\text{m}}{\text{s}^2}$
a_{x}	x-component of acceleration	$\frac{\text{m}}{\text{s}^2}$
${a_{\text{x}}}^{\text{c}}$	x-component of constant acceleration	$\frac{\text{m}}{\text{s}^2}$
a_{y}	y-component of acceleration	$\frac{\text{m}}{\text{s}^2}$
${a_{\text{y}}}^{\text{c}}$	y-component of constant acceleration	$\frac{\text{m}}{\text{s}^2}$
$\mathbf{a}(t)$	Acceleration	$\frac{\text{m}}{\text{s}^2}$
$\mathbf{a}^{\text{c}}$	Constant acceleration vector	$\frac{\text{m}}{\text{s}^2}$
d_{offset}	Distance between the target position and the landing position	m
g	Magnitude of gravitational acceleration	$\frac{\text{m}}{\text{s}^2}$
p	Scalar position	m

Continued on next page

Table 2: Table of Symbols (Continued)

Symbol	Description	Units
$p(t)$	1D position	m
p^{i}	Initial position	m
p_{land}	Landing position	m
p_{target}	Target position	m
p_{x}	x-component of position	m
${p_{\text{x}}}^{\text{i}}$	x-component of initial position	m
p_{y}	y-component of position	m
${p_{\text{y}}}^{\text{i}}$	y-component of initial position	m
$\mathbf{p}(t)$	Position	m
s	Output message as a string	–
t	Time	s
t_{flight}	Flight duration	s
v	Speed	$\frac{\text{m}}{\text{s}}$
$v(t)$	1D speed	$\frac{\text{m}}{\text{s}}$
v^{i}	Initial speed	$\frac{\text{m}}{\text{s}}$
v_{launch}	Launch speed	$\frac{\text{m}}{\text{s}}$
v_{x}	x-component of velocity	$\frac{\text{m}}{\text{s}}$
${v_{\text{x}}}^{\text{i}}$	x-component of initial velocity	$\frac{\text{m}}{\text{s}}$
v_{y}	y-component of velocity	$\frac{\text{m}}{\text{s}}$
${v_{\text{y}}}^{\text{i}}$	y-component of initial velocity	$\frac{\text{m}}{\text{s}}$
$\mathbf{v}(t)$	Velocity	$\frac{\text{m}}{\text{s}}$
$\mathbf{v}^{\text{i}}$	Initial velocity	$\frac{\text{m}}{\text{s}}$
ε	Hit tolerance	–
θ	Launch angle	rad
π	Ratio of circumference to diameter for any circle	–

1.3 Abbreviations and Acronyms

Table 3: Abbreviations and Acronyms

Abbreviation	**Full Form**
1D	One-Dimensional
2D	Two-Dimensional
A	Assumption
DD	Data Definition
GD	General Definition
GS	Goal Statement
IM	Instance Model
PS	Physical System Description
R	Requirement
RefBy	Referenced by
Refname	Reference Name
SRS	Software Requirements Specification
TM	Theoretical Model
Uncert.	Typical Uncertainty

2 Introduction

Projectile motion is a common problem in physics. Therefore, it is useful to have a program to solve and model these types of problems. Common examples of projectile motion include ballistics problems (missiles, bullets, etc.) and the flight of balls in various sports (baseball, golf, football, etc.). The program documented here is called Projectile.

The following section provides an overview of the Software Requirements Specification (SRS) for Projectile. This section explains the purpose of this document, the scope of the requirements, the characteristics of the intended reader, and the organization of the document.

2.1 Purpose of Document

The primary purpose of this document is to record the requirements of Projectile. Goals, assumptions, theoretical models, definitions, and other model derivation information are specified, allowing the reader to fully understand and verify the purpose and scientific basis of Projectile. With the exception of system constraints, this SRS will remain abstract, describing what problem is being solved, but not how to solve it.

This document will be used as a starting point for subsequent development phases, including writing the design specification and the software verification and validation plan. The design document will show how the requirements are to be realized, including decisions on the numerical algorithms and programming environment. The verification and validation plan will show the steps that will be used to increase confidence in the software documentation and the implementation. Although the SRS fits in a series of documents that follow the so-called waterfall model, the actual development process is not constrained in any way. Even when the waterfall model is not followed, as Parnas and Clements point out [**parnasClements1986**], the most logical way to present the documentation is still to "fake" a rational design process.

2.2 Scope of Requirements

The scope of the requirements includes the analysis of a two-dimensional (2D) projectile motion problem with constant acceleration.

[We are only interested in the position of the projectile, not its orientation A:noOrientation. —SS]

[We assume that forces are not relevant for the model so that we only need kinematic equations A:kinematicOnly. —SS]

2.3 Characteristics of Intended Reader

Reviewers of this documentation should have an understanding of undergraduate level 1 physics and undergraduate level 1 calculus. The users of Projectile can have a lower level of expertise, as explained in Sec:User Characteristics.

2.4 Organization of Document

The organization of this document follows the template for an SRS for scientific computing software proposed by [**koothoor2013**], [**smithLai2005**], [**smithEtAl2007**], and [**smithKoothoor2016**]. The presentation follows the standard pattern of presenting goals, theories, definitions, and assumptions. For readers that would like a more bottom up approach, they can start reading the instance models and trace back to find any additional information they require.

The goal statements are refined to the theoretical models and the theoretical models to the instance models.

3 General System Description

This section provides general information about the system. It identifies the interfaces between the system and its environment, describes the user characteristics, and lists the system constraints.

Figure 1: System Context

3.1 System Context

Fig:sysCtxDiag shows the system context. A circle represents an entity external to the software, the user in this case. A rectangle represents the software system itself (Projectile). Arrows are used to show the data flow between the system and its environment.

The interaction between the product and the user is through an application programming interface. The responsibilities of the user and the system are as follows:

- User Responsibilities
  - Provide initial conditions of the physical state of the motion and the input data related to the Projectile, ensuring no errors in the data entry.
  - Ensure that consistent units are used for input variables.
  - Ensure required software assumptions are appropriate for any particular problem input to the software.
- Projectile Responsibilities
  - Detect data type mismatch, such as a string of characters input instead of a floating point number.
  - Determine if the inputs satisfy the required physical and software constraints.
  - Calculate the required outputs.

3.2 User Characteristics

The end user of Projectile should have an understanding of high school physics and high school calculus.

3.3 System Constraints

There are no system constraints.

4 Specific System Description

This section first presents the problem description, which gives a high-level view of the problem to be solved. This is followed by the solution characteristics specification, which presents the assumptions, theories, and definitions that are used.

4.1 Problem Description

A system is needed to predict whether a launched projectile hits its target.

4.1.1 Terminology and Definitions

This subsection provides a list of terms that are used in the subsequent sections and their meaning, with the purpose of reducing ambiguity and making it easier to correctly understand the requirements.

- Launcher: Where the projectile is launched from and the device that does the launching.
- Projectile: The object to be launched at the target.
- Target: Where the projectile should be launched to.
- Gravity: The force that attracts one physical body with mass to another.
- Cartesian coordinate system: A coordinate system that specifies each point uniquely in a plane by a set of numerical coordinates, which are the signed distances to the point from two fixed perpendicular oriented lines, measured in the same unit of length (from [2]).
- Rectilinear: Occurring [fixed typo in "Ocurring" —SS] in one dimension.

4.1.2 Physical System Description

The physical system of Projectile, as shown in Fig:Launch, includes the following elements:

PS1: The launcher.

PS2: The projectile (with initial velocity $\mathbf{v}^{\mathrm{i}}$ and launch angle θ).

PS3: The target.

4.1.3 Goal Statements

Given the initial velocity vector of the projectile and the geometric layout of the launcher and target, the goal statement is:

targetHit: Determine if the projectile hits the target.

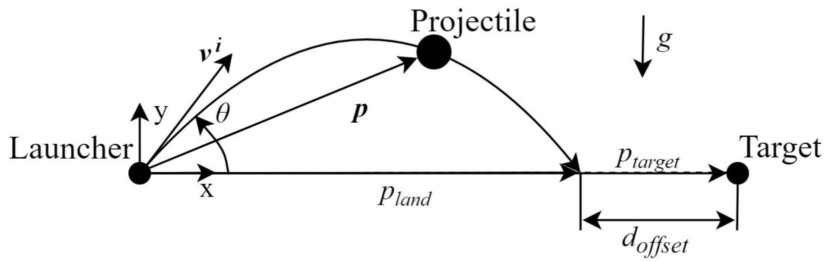

Figure 2: The physical system

4.2 Solution Characteristics Specification

The instance models that govern Projectile are presented in the Instance Model Section. The information to understand the meaning of the instance models and their derivation is also presented, so that the instance models can be verified.

4.2.1 [Types —SS]

[time = $\mathbb{R}$ —SS]

4.2.2 [Scope Assumptions —SS]

This section simplifies the original problem and helps in developing the theoretical models by filling in the missing information for the physical system. The assumptions refine the scope by providing more detail.

noOrientation: [The orientation of the projectile is ignored. We only care about its translation not its rotation. (RefBy: TM:acceleration and TM:velocity.) —SS]

kinematicOnly: [The motion of the projectile is modelled with only kinematic equations. Forces are not considered. —SS]

4.2.3 [Background Theory Assumptions —SS]

cartSyst: A Cartesian coordinate system is used (from A:neglectCurv). (RefBy: GD:velVec and GD:posVec.)

4.2.4 [Helper Theory Assumptions (GD:rectVel, GD:rectPos) —SS]

oneD: The motion of the particle is one dimensional. (RefBy: GD:velVec and GD:posVec.)

constAccel: The acceleration is constant (from A:accelXZero, A:accelYGravity, A:neglectDrag, and A:freeFlight). (RefBy: GD:velVec and GD:posVec.)

4.2.5 [Theory Assumptions (GD:velVec, GD:posVec) —SS]

twoD: The variables only depend on two-dimensions (2D). (RefBy: GD:velVec and GD:posVec.) [changed twoDMotion to just twoD, so that it can be used for things that are 2D other than just the motion. —SS]

constAccelX: [The acceleration is constant in the x direction. (RefBy: GD:velVec and GD:posVec.) —SS]

constAccelY: [The acceleration is constant in the y direction. (RefBy: GD:velVec and GD:posVec.) —SS]

4.2.6 [Theory Assumptions —SS]

twoD: The projectile motion is two-dimensional (2D). (RefBy: GD:velVec and GD:posVec.)

yAxisGravity: The direction of the y-axis is directed opposite to gravity. (RefBy: IM:calOfLandingDist, IM:calOfLandingTime, and A:accelYGravity.)

launchOrigin: The launcher is coincident with the origin. (RefBy: IM:calOfLandingDist and IM:calOfLandingTime.)

targetXAxis: The target lies on the x-axis (from A:neglectCurv). (RefBy: IM:calOfLandingTime.)

posXDirection: The positive x-direction is from the launcher to the target. (RefBy: IM:offsetIM, IM:messageIM, IM:calOfLandingDist, and IM:calOfLandingTime.)

accelXZero: The acceleration in the x-direction is zero. (RefBy: IM:calOfLandingDist and A:constAccel.)

accelYGravity: The acceleration in the y-direction is the acceleration due to gravity (from A:yAxisGravity). (RefBy: IM:calOfLandingTime and A:constAccel.)

neglectDrag: Air drag is neglected. (RefBy: A:constAccel.)

pointMass: The size and shape of the projectile are negligible, so that it can be modelled as a point mass. (RefBy: GD:rectVel and GD:rectPos.)

freeFlight: The flight is free; there are no collisions during the trajectory of the projectile. (RefBy: A:constAccel.)

neglectCurv: The distance is small enough that the curvature of the celestial body can be neglected. (RefBy: A:targetXAxis and A:cartSyst.)

timeStartZero: Time starts at zero. (RefBy: GD:velVec, GD:rectVel, GD:rectPos, GD:posVec, and IM:calOfLandingTime.)

gravAccelValue: The acceleration due to gravity is assumed to have the value provided in the section for Values of Auxiliary Constants. (RefBy: IM:calOfLandingDist and IM:calOfLandingTime.)

4.2.7 [Final Theory Assumptions —SS]

twoD: The projectile motion is two-dimensional (2D). (RefBy: GD:velVec and GD:posVec.)

yAxisGravity: The direction of the y-axis is directed opposite to gravity. (RefBy: IM:calOfLandingDist, IM:calOfLandingTime, and A:accelYGravity.)

launchOrigin: The launcher is coincident with the origin. (RefBy: IM:calOfLandingDist and IM:calOfLandingTime.)

targetXAxis: The target lies on the x-axis (from A:neglectCurv). (RefBy: IM:calOfLandingTime.)

posXDirection: The positive x-direction is from the launcher to the target. (RefBy: IM:offsetIM, IM:messageIM, IM:calOfLandingDist, and IM:calOfLandingTime.)

accelXZero: The acceleration in the x-direction is zero. (RefBy: IM:calOfLandingDist and A:constAccel.)

accelYGravity: The acceleration in the y-direction is the acceleration due to gravity (from A:yAxisGravity). (RefBy: IM:calOfLandingTime and A:constAccel.)

neglectDrag: Air drag is neglected. (RefBy: A:constAccel.)

pointMass: The size and shape of the projectile are negligible, so that it can be modelled as a point mass. (RefBy: GD:rectVel and GD:rectPos.)

freeFlight: The flight is free; there are no collisions during the trajectory of the projectile. (RefBy: A:constAccel.)

neglectCurv: The distance is small enough that the curvature of the celestial body can be neglected. (RefBy: A:targetXAxis and A:cartSyst.)

timeStartZero: Time starts at zero. (RefBy: GD:velVec, GD:rectVel, GD:rectPos, GD:posVec, and IM:calOfLandingTime.)

gravAccelValue: The acceleration due to gravity is assumed to have the value provided in the section for Values of Auxiliary Constants. (RefBy: IM:calOfLandingDist and IM:calOfLandingTime.)

4.2.8 [Context Theories —SS]

[Some theories do not have to be explicitly invoked. They are part of the context for the other theories, without having to be explicitly stated or defined. The context theories for this problem are as follows:

- real arithmetic
- vectors

- Cartesian coordinate system
- differentiation
- integration

—SS]

4.2.9 [Background Theories (BT) —SS]

This section focuses on the general equations and laws that Projectile is based on. [Maybe relabel all of the background theories with the prefix BT, instead of TM? —SS]

Refname	**TM:acceleration**
Label	Acceleration
[Input —SS]	[$\mathbf{v}$: time $\rightarrow \mathbb{R}^3$ —SS]
[Output —SS]	[$\mathbf{a}$: time $\rightarrow \mathbb{R}^3$ —SS]
[Input Constraints —SS]	[None —SS]
[Output Constraints —SS]	[None —SS]
Equation	$$\mathbf{a}(t) = \frac{d\mathbf{v}(t)}{dt}$$
Description	$\mathbf{a}(t)$ is the acceleration ($\frac{\text{m}}{\text{s}^2}$) t is the time (s) $\mathbf{v}(t)$ is the velocity ($\frac{\text{m}}{\text{s}}$)
Notes	The velocity and acceleration of the body are expressed using a Cartesian coordinate system (A:cartSyst). That is, the coordinate system is rectangular (orthonormal).
Source	[1]
RefBy	GD:rectVel

Preconditions for TM:acceleration

- A:cartSyst

Refname	**TM:velocity**
Label	Velocity
[Input —SS]	[$\mathbf{p}$: time $\rightarrow \mathbb{R}^3$ —SS]
[Output —SS]	[$\mathbf{v}$: time $\rightarrow \mathbb{R}^3$ —SS]
[Input Constraints —SS]	[None —SS]
[Output Constraints —SS]	[None —SS]
Equation	$$\mathbf{v}(t) = \frac{d\mathbf{p}(t)}{dt}$$
Description	$\mathbf{v}(t)$ is the velocity ($\frac{\text{m}}{\text{s}}$) t is the time (s) $\mathbf{p}(t)$ is the position (m)
Notes	The position and velocity of the body are expressed using a Cartesian coordinate system (A:cartSyst). That is, the coordinate system is rectangular (orthonormal).
Source	[3]
RefBy	GD:rectPos

Preconditions for TM:velocity

- A:cartSyst

Refname	TM:directionCosines
Label	[Direction Cosines Representation for Vectors —SS]
[Input —SS]	[$\alpha : \mathbb{R}, \beta : \mathbb{R}, \gamma : \mathbb{R}, \lvert\mathbf{b}\rvert : \mathbb{R}$ —SS]
[Output —SS]	[$b_x : \mathbb{R}, b_y : \mathbb{R}, b_z : \mathbb{R}$ —SS]
[Input Constraints —SS]	[$\cos^2(\alpha) + \cos^2(\beta) + \cos^2(\gamma) = 1$ —SS]
[Output Constraints —SS]	[None —SS]
Equation	$$b_x = \lvert\mathbf{b}\rvert \cos(\alpha), b_y = \lvert\mathbf{b}\rvert \cos(\beta), b_z = \lvert\mathbf{b}\rvert \cos(\gamma)$$
Description	α is the angle between the vector and the positive x axis β is the angle between the vector and the positive y axis γ is the angle between the vector and the positive z axis $\lvert\mathbf{b}\rvert$ is the magnitude of the vector b_x is the x component of the vector $\mathbf{b}$ b_y is the y component of the vector $\mathbf{b}$ b_z is the z component of the vector $\mathbf{b}$
Notes	[The vector $\mathbf{b}$ is in a three dimensional Cartesian coordinate system (A:cartSyst). —SS] [A figure showing the angles for a sample vector would be a nice addition. —SS]
Source	[**<empty citation>**] web-page resource, Long book
RefBy	D:magAngleToCompRep]

Preconditions for TM:directionCosines

- A:cartSyst

4.2.10 [Helper Theories (GD:rectVel and GD:rectPos) —SS]

This section collects the laws and equations that will be used to build the instance models. [We should remove the prefix GD. Maybe we should replace it with the prefix TM? —SS]

Refname	GD:rectVel
Label	Rectilinear (1D) velocity as a function of time for constant acceleration
Units	$\frac{\text{m}}{\text{s}}$ [Do we need units as a separate field? user option? —SS]
[Input —SS]	[$v^{\text{i}} : \mathbb{R}, a^c : \mathbb{R}, t :$ time —SS]
[Output —SS]	[$v : \mathbb{R}$ —SS]
[Input Constraints —SS]	[$t \geq 0$ —SS] [A:timeStartZero —SS]
[Output Constraints —SS]	[None —SS]
Equation	$$v(t) = v^{\text{i}} + a^c t$$
Description	$v(t)$ is the 1D speed ($\frac{\text{m}}{\text{s}}$) v^{i} is the initial speed ($\frac{\text{m}}{\text{s}}$) a^c is the constant acceleration ($\frac{\text{m}}{\text{s}^2}$) t is the time (s)
[Notes —SS]	[See detailed derivation below —SS]
Source	[4, (pg. 8)]
RefBy	GD:velVec and GD:rectPos

Detailed derivation of rectilinear velocity: [We start from the theory of acceleration TM:acceleration for a body in 3D Cartesian space: —SS]

$$\mathbf{a}(t) = \frac{d\mathbf{v}(t)}{dt}$$

[At this point we assume the body only travels in a straight line along one dimension of the coordinate system (A:oneD): —SS]

$$a(t) = \frac{dv(t)}{dt}$$

[We now assume that the acceleration is constant (does not vary with time) (A:constAccel) represented by a^c. The initial velocity (at $t = 0$, from A:timeStartZero) is represented by v^{i}. We now have: —SS]

$$a^c = \frac{dv}{dt}$$

Rearranging and integrating, we have:

$$\int_{v^{\mathrm{i}}}^{v} 1\, dv = \int_0^t a^c\, dt$$

Performing the integration, we have the required equation:

$$v(t) = v^{\mathrm{i}} + a^c t$$

[Preconditions for GD:rectVel —SS]

- A:cartSyst (inherited from TM:acceleration)
- A:oneD
- A:constAccel
- A:timeStartZero

Refname	GD:rectPos
Label	Rectilinear (1D) position as a function of time for constant acceleration
Units	m
[Input —SS]	[$p^{\text{i}} : \mathbb{R}, v^{\text{i}} : \mathbb{R}, a^c : \mathbb{R}, t :$ time —SS]
[Output —SS]	[$p : \mathbb{R}$ —SS]
[Input Constraints —SS]	[$t \geq 0$ —SS] [A:timeStartZero —SS]
[Output Constraints —SS]	[None —SS]
Equation	$$p(t) = p^{\text{i}} + v^{\text{i}}t + \frac{a^c t^2}{2}$$
Description	$p(t)$ is the 1D position (m) p^{i} is the initial position (m) v^{i} is the initial speed ($\frac{\text{m}}{\text{s}}$) t is the time (s) a^c is the constant acceleration ($\frac{\text{m}}{\text{s}^2}$)
[Notes —SS]	[See detailed derivation below —SS]
Source	[4, (pg. 8)]
RefBy	GD:posVec

Detailed derivation of rectilinear position: [We start from the kinematic equation for velocity TM:velocity for a body in 3D Cartesian space: —SS]

$$\mathbf{v}(t) = \frac{d\mathbf{p}(t)}{dt}$$

[At this point we assume the body only travels in a straight line along one dimension of the coordinate system (A:oneD): —SS]

$$v(t) = \frac{dp(t)}{dt}$$

[The initial position (at $t = 0$, from A:timeStartZero) is represented by p^{i}. Rearranging the above equation and integrating we have: —SS]

$$\int_{p^{\mathrm{i}}}^{p(t)} 1\, dp = \int_0^t v(t)\, dt$$

[This equation has been changed from the previous version to show $p(t)$ and $v(t)$ —SS]

[We now assume that the acceleration is constant (does not vary with time) (A:constAccel) represented by a^c. The initial velocity (at $t = 0$, from A:timeStartZero) is represented by v^{i}. Since we satisfy the preconditions for GD:rectVel we can replace $v(t)$ to get: —SS]

$$\int_{p^{\mathrm{i}}}^{p(t)} 1\, dp = \int_0^t v^{\mathrm{i}} + a^c t\, dt$$

[The above equation has been changed from the previous version to show $p(t)$. —SS] Performing the integration, we have the required equation:

$$p(t) = p^{\mathrm{i}} + v^{\mathrm{i}} t + \frac{a^c t^2}{2}$$

Preconditions for GD:rectPos

- A:cartSyst (inherited from TM:velocity)
- A:oneD
- A:constAccel
- A:timeStartZero

4.2.11 [Theories (GD:velVec and GD:posVec) —SS]

Refname	GD:velVec
Label	Velocity vector as a function of time for 2D motion under constant acceleration
Units	$\frac{\text{m}}{\text{s}}$
[Input —SS]	[$v_{\text{x}}^{\text{i}} : \mathbb{R}, v_{\text{y}}^{\text{i}} : \mathbb{R}, {a_{\text{x}}}^{\text{c}} : \mathbb{R}, {a_{\text{y}}}^{\text{c}} : \mathbb{R}, t$: time —SS]
[Output —SS]	[$\mathbf{v} : \mathbb{R}^2$ —SS]
[Input Constraints —SS]	[$t \geq 0$ —SS] [A:timeStartZero —SS]
[Output Constraints —SS]	[None —SS]
Equation	$$\mathbf{v}(t) = \begin{bmatrix} {v_{\text{x}}}^{\text{i}} + {a_{\text{x}}}^{\text{c}}t \\ {v_{\text{y}}}^{\text{i}} + {a_{\text{y}}}^{\text{c}}t \end{bmatrix}$$
Description	$\mathbf{v}(t)$ is the velocity ($\frac{\text{m}}{\text{s}}$) ${v_{\text{x}}}^{\text{i}}$ is the x-component of initial velocity ($\frac{\text{m}}{\text{s}}$) ${a_{\text{x}}}^{\text{c}}$ is the x-component of constant acceleration ($\frac{\text{m}}{\text{s}^2}$) t is the time (s) ${v_{\text{y}}}^{\text{i}}$ is the y-component of initial velocity ($\frac{\text{m}}{\text{s}}$) ${a_{\text{y}}}^{\text{c}}$ is the y-component of constant acceleration ($\frac{\text{m}}{\text{s}^2}$)
Source	–
RefBy	

Detailed derivation of velocity vector: For a two-dimensional Cartesian coordinate system (A:twoD and A:cartSyst), we can represent the velocity vector as $\mathbf{v}(t) = \begin{bmatrix} v_{\text{x}}([t---SS]) \\ v_{\text{y}}([t---SS]) \end{bmatrix}$ and the acceleration vector as $\mathbf{a}(t) = \begin{bmatrix} a_{\text{x}}([t---SS]) \\ a_{\text{y}}([t---SS]) \end{bmatrix}$. The acceleration is assumed to be constant in both the x direction ${a_x}^c$ (A:constAccelX) and y direction ${a_y}^c$ (A:constAccelY). The constant acceleration vector is represented as $\mathbf{a}^c = \begin{bmatrix} {a_x}^c \\ {a_y}^c \end{bmatrix}$. The initial velocity (at $t = 0$, from A:timeStartZero) is represented by $\mathbf{v}^i = \begin{bmatrix} {v_x}^i \\ {v_y}^i \end{bmatrix}$. [For each coordinate direction we satisfy the preconditions for GD:rectVel (A:cartSyst, A:oneD, A:constAccel, A:timeStartZero). This means we can use the one dimensional equation for each of the two coordinate directions to yield the required equation: —SS]

$$\mathbf{v}(t) = \begin{bmatrix} {v_x}^i + {a_x}^c t \\ {v_y}^i + {a_y}^c t \end{bmatrix}$$

Preconditions for GD:velVec

- A:timeStartZero (inherited from GD:rectVel)
- A:cartSyst (inherited from GD:rectVel)
- A:twoD (A:oneD in both the x and y directions)
- A:constAccelX (A:constAccel in x direction)
- A:constAccelY (A:constAccel in y direction)

Refname	**GD:posVec**
Label	Position vector as a function of time for 2D motion under constant acceleration
Units	m
[Input —SS]	[$p_{\mathrm{x}}^{\mathrm{i}} : \mathbb{R}, v_{\mathrm{x}}^{\mathrm{i}} : \mathbb{R}, p_{\mathrm{y}}^{\mathrm{i}} : \mathbb{R}, v_{\mathrm{y}}^{\mathrm{i}} : \mathbb{R}, {a_{\mathrm{x}}}^{\mathrm{c}} : \mathbb{R}, {a_{\mathrm{y}}}^{\mathrm{c}} : \mathbb{R}, t : \mathrm{time}$ —SS]
[Output —SS]	[$\mathbf{p} : \mathbb{R}^2$ —SS]
[Input Constraints —SS]	[$t \geq 0$ —SS] [A:timeStartZero —SS]
[Output Constraints —SS]	[None —SS]
Equation	$$\mathbf{p}(t) = \begin{bmatrix} {p_{\mathrm{x}}}^{\mathrm{i}} + {v_{\mathrm{x}}}^{\mathrm{i}}t + \frac{{a_{\mathrm{x}}}^{\mathrm{c}}t^2}{2} \\ {p_{\mathrm{y}}}^{\mathrm{i}} + {v_{\mathrm{y}}}^{\mathrm{i}}t + \frac{{a_{\mathrm{y}}}^{\mathrm{c}}t^2}{2} \end{bmatrix}$$
Description	$\mathbf{p}(t)$ is the position (m) ${p_{\mathrm{x}}}^{\mathrm{i}}$ is the x-component of initial position (m) ${v_{\mathrm{x}}}^{\mathrm{i}}$ is the x-component of initial velocity ($\frac{\mathrm{m}}{\mathrm{s}}$) t is the time (s) ${a_{\mathrm{x}}}^{\mathrm{c}}$ is the x-component of constant acceleration ($\frac{\mathrm{m}}{\mathrm{s}^2}$) ${p_{\mathrm{y}}}^{\mathrm{i}}$ is the y-component of initial position (m) ${v_{\mathrm{y}}}^{\mathrm{i}}$ is the y-component of initial velocity ($\frac{\mathrm{m}}{\mathrm{s}}$) ${a_{\mathrm{y}}}^{\mathrm{c}}$ is the y-component of constant acceleration ($\frac{\mathrm{m}}{\mathrm{s}^2}$)
Source	–
RefBy	IM:calOfLandingDist and IM:calOfLandingTime

Detailed derivation of position vector: For a two-dimensional Cartesian coordinate system (A:twoD and A:cartSyst), we can represent the position vector as $\mathbf{p}(t) = \begin{bmatrix} p_{\mathrm{x}}[(t) - - - SS] \\ p_{\mathrm{y}}[(t) - - - SS] \end{bmatrix}$, the velocity vector as $\mathbf{v}(t) = \begin{bmatrix} v_{\text{x}} \\ v_{\text{y}} \end{bmatrix}$, and the acceleration vector as $\mathbf{a}(t) = \begin{bmatrix} a_{\text{x}}[(t) - - - SS] \\ a_{\text{y}}[(t) - - - SS] \end{bmatrix}$. The acceleration is assumed to be constant (A:constAccel) and the constant acceleration vector is represented as $\mathbf{a}^{\text{c}} = \begin{bmatrix} {a_{\text{x}}}^{\text{c}} \\ {a_{\text{y}}}^{\text{c}} \end{bmatrix}$. The initial velocity (at $t = 0$, from A:timeStartZero) is represented by $\mathbf{v}^{\text{i}} = \begin{bmatrix} {v_{\text{x}}}^{\text{i}} \\ {v_{\text{y}}}^{\text{i}} \end{bmatrix}$. [For each coordinate direction we satisfy the preconditions for GD:rectPos (A:cartSyst, A:oneD, A:constAccel, A:timeStartZero). This means we can use the one dimensional equation for each of the two coordinate directions to yield the required equation: —SS]

$$\mathbf{p}(t) = \begin{bmatrix} {p_{\text{x}}}^{\text{i}} + {v_{\text{x}}}^{\text{i}} t + \frac{{a_{\text{x}}}^{\text{c}} t^2}{2} \\ {p_{\text{y}}}^{\text{i}} + {v_{\text{y}}}^{\text{i}} t + \frac{{a_{\text{y}}}^{\text{c}} t^2}{2} \end{bmatrix}$$

Preconditions for GD:posVec

- A:timeStartZero (inherited from GD:rectPos)
- A:cartSyst (inherited from GD:rectPos)
- A:twoD (A:oneD in both the x and y directions)
- A:constAccelX (A:constAccel in x direction)
- A:constAccelY (A:constAccel in y direction)

Refname	GD:magAngleToCompRep
Label	[Conversion of Magnitude and Angle Representation of a Vector to the Component Representation —SS]
[Input —SS]	[$\theta : \mathbb{R}, \lvert\mathbf{b}\rvert : \mathbb{R}$ —SS]
[Output —SS]	[$b_x : \mathbb{R}, b_y : \mathbb{R}$ —SS]
[Input Constraints —SS]	[None —SS]
[Output Constraints —SS]	[None —SS]
Equation	$$b_x = \lvert\mathbf{b}\rvert \cos(\theta), b_y = \lvert\mathbf{b}\rvert \sin(\theta)$$
Description	θ is the angle between the vector and the positive x axis $\lvert\mathbf{b}\rvert$ is the magnitude of the vector b_x is the x component of the vector $\mathbf{b}$ b_y is the y component of the vector $\mathbf{b}$
Notes	[The vector $\mathbf{b}$ is in a two dimensional (A:twoD) Cartesian coordinate system (A:cartSyst). —SS] [The equations can be derived from TM:directionCosines for a 2D sytem. In a 2D system, the angle γ is not relevant. The angle will be $\pi/2$, so the $\cos(\gamma) = \cos(\pi/2) = 0$. In the 2D case we rename the angle α as θ. The angle β is related to θ by $\beta = \pi/2 - \theta$; therefore, $\cos(\beta) = \cos(\pi/2 - \theta) = \sin(\theta)$. —SS]
Source	[**<empty citation>**]
RefBy	DD:speedIX, DD:speedIY

Preconditions for GD:magAngleToCompRep

- A:cartSyst (inherited from directionCosines)
- A:twoD

4.2.12 Data Definitions

This section collects and defines all the data needed to build the instance models.

Refname	**DD:vecMag** [REMOVE —SS]
Label	Speed

Refname	**DD:speedIX**
Label	x-component of initial velocity
Symbol	${v_{\text{x}}}^{\text{i}}$
Units	$\frac{\text{m}}{\text{s}}$
Equation	$${v_{\text{x}}}^{\text{i}} = v^{\text{i}} \cos\left(\theta\right)$$
Description	${v_{\text{x}}}^{\text{i}}$ is the x-component of initial velocity ($\frac{\text{m}}{\text{s}}$) v^{i} is the initial speed ($\frac{\text{m}}{\text{s}}$) θ is the launch angle (rad)
Notes	[This equation is a relabelling of GD:magAngleToCompRep—v^{i} is $\|\mathbf{b}\|$, ${v_{\text{x}}}^{\text{i}}$ is b_x and θ is θ. —SS] θ is shown in Fig:Launch.
Source	–
RefBy	IM:calOfLandingDist

Refname	DD:speedIY
Label	y-component of initial velocity
Symbol	${v_{\text{y}}}^{\text{i}}$
Units	$\frac{\text{m}}{\text{s}}$
Equation	$${v_{\text{y}}}^{\text{i}} = v^{\text{i}} \sin\left(\theta\right)$$
Description	${v_{\text{y}}}^{\text{i}}$ is the y-component of initial velocity ($\frac{\text{m}}{\text{s}}$) v^{i} is the initial speed ($\frac{\text{m}}{\text{s}}$) θ is the launch angle (rad)
Notes	[This equation is a relabelling of GD:magAngleToCompRep—v^{i} is $\lvert\mathbf{b}\rvert$, ${v_{\text{y}}}^{\text{i}}$ is b_y and θ is θ. —SS] θ is shown in Fig:Launch.
Source	–
RefBy	IM:calOfLandingTime

4.2.13 Instance Models

This section transforms the problem defined in the problem description into one which is expressed in mathematical terms. It uses concrete symbols defined in the data definitions to replace the abstract symbols in the models identified in theoretical models and general definitions.

Refname	IM:calOfLandingTime
Label	Calculation of landing time
Input	v_{launch} [:ℝ —SS], θ [:ℝ —SS]
Output	t_{flight}
Input Constraints	$$v_{\text{launch}} > 0$$ $$0 < \theta < \frac{\pi}{2}$$
Output Constraints	$$t_{\text{flight}} > 0$$
Equation	$$t_{\text{flight}} = \frac{2v_{\text{launch}} \sin(\theta)}{g}$$
Description	t_{flight} is the flight duration (s) v_{launch} is the launch speed ($\frac{\text{m}}{\text{s}}$) θ is the launch angle (rad) g is the magnitude of gravitational acceleration ($\frac{\text{m}}{\text{s}^2}$)
Notes	The constraint $0 < \theta < \frac{\pi}{2}$ is from A:posXDirection and A:yAxisGravity, and is shown in Fig:Launch. g is defined in A:gravAccelValue. The constraint $t_{\text{flight}} > 0$ is from A:timeStartZero.
Source	–
RefBy	IM:calOfLandingDist, FR:Output-Values, and FR:Calculate-Values

Detailed derivation of flight duration: We know that ${p_{\text{y}}}^{\text{i}} = 0$ (A:launchOrigin) and ${a_{\text{y}}}^{\text{c}} = -g$ (A:accelYGravity). Substituting these values into the y-direction of GD:posVec gives us:

$$p_{\text{y}} = {v_{\text{y}}}^{\text{i}} t - \frac{g t^2}{2}$$

To find the time that the projectile lands, we want to find the t value (t_{flight}) where $p_{\text{y}} = 0$ (since the target is on the x-axis from A:targetXAxis). From the equation above we get:

$${v_{\text{y}}}^{\text{i}} t_{\text{flight}} - \frac{g {t_{\text{flight}}}^2}{2} = 0$$

Dividing by t_{flight} (with the constraint $t_{\text{flight}} > 0$) gives us:

$${v_{\text{y}}}^{\text{i}} - \frac{g t_{\text{flight}}}{2} = 0$$

Solving for t_{flight} gives us:

$$t_{\text{flight}} = \frac{2 {v_{\text{y}}}^{\text{i}}}{g}$$

From DD:speedIY (with $v^{\text{i}} = v_{\text{launch}}$) we can replace ${v_{\text{y}}}^{\text{i}}$:

$$t_{\text{flight}} = \frac{2 v_{\text{launch}} \sin(\theta)}{g}$$

Preconditions for IM:calOfLandingTime

-

Refname	IM:calOfLandingDist
Label	Calculation of landing position
Input	v_{launch}, θ
Output	p_{land}
Input Constraints	$v_{\text{launch}} > 0$ $0 < \theta < \frac{\pi}{2}$
Output Constraints	$p_{\text{land}} > 0$
Equation	$p_{\text{land}} = \frac{2{v_{\text{launch}}}^2 \sin(\theta) \cos(\theta)}{g}$
Description	p_{land} is the landing position (m) v_{launch} is the launch speed ($\frac{\text{m}}{\text{s}}$) θ is the launch angle (rad) g is the magnitude of gravitational acceleration ($\frac{\text{m}}{\text{s}^2}$)
Notes	The constraint $0 < \theta < \frac{\pi}{2}$ is from A:posXDirection and A:yAxisGravity, and is shown in Fig:Launch. g is defined in A:gravAccelValue. The constraint $p_{\text{land}} > 0$ is from A:posXDirection.
Source	–
RefBy	IM:offsetIM and FR:Calculate-Values

Detailed derivation of landing position: We know that ${p_{\text{x}}}^{\text{i}} = 0$ (A:launchOrigin) and ${a_{\text{x}}}^{\text{c}} = 0$ (A:accelXZero). Substituting these values into the x-direction of GD:posVec gives us:

$$p_{\text{x}} = {v_{\text{x}}}^{\text{i}} t$$

To find the landing position, we want to find the p_{x} value (p_{land}) at flight duration (from IM:calOfLandingTime):

$$p_{\text{land}} = \frac{{v_{\text{x}}}^{\text{i}} \cdot 2 v_{\text{launch}} \sin\left(\theta\right)}{g}$$

From DD:speedIX (with $v^{\text{i}} = v_{\text{launch}}$) we can replace ${v_{\text{x}}}^{\text{i}}$:

$$p_{\text{land}} = \frac{v_{\text{launch}} \cos\left(\theta\right) \cdot 2 v_{\text{launch}} \sin\left(\theta\right)}{g}$$

Rearranging this gives us the required equation:

$$p_{\text{land}} = \frac{2 {v_{\text{launch}}}^{2} \sin\left(\theta\right) \cos\left(\theta\right)}{g}$$

Preconditions for IM:calOfLandingDistDeriv

-

Refname	IM:offsetIM
Label	Offset
Input	p_{land}, p_{target}
Output	d_{offset}
Input Constraints	$$p_{\text{land}} > 0$$ $$p_{\text{target}} > 0$$
Output Constraints	
Equation	$$d_{\text{offset}} = p_{\text{land}} - p_{\text{target}}$$
Description	d_{offset} is the distance between the target position and the landing position (m) p_{land} is the landing position (m) p_{target} is the target position (m)
Notes	p_{land} is from IM:calOfLandingDist. The constraints $p_{\text{land}} > 0$ and $p_{\text{target}} > 0$ are from A:posXDirection.
Source	–
RefBy	IM:messageIM, FR:Output-Values, and FR:Calculate-Values

Preconditions for IM:offsetIM

-

Refname	IM:messageIM
Label	Output message
Input	d_{offset}, p_{target}
Output	s
Input Constraints	$d_{\text{offset}} > -p_{\text{target}}$ $p_{\text{target}} > 0$
Output Constraints	
Equation	$s = \begin{cases} \text{“The target was hit.”}, & \lvert\frac{d_{\text{offset}}}{p_{\text{target}}}\rvert < \varepsilon \\ \text{“The projectile fell short.”}, & d_{\text{offset}} < 0 \\ \text{“The projectile went long.”}, & d_{\text{offset}} > 0 \end{cases}$
Description	s is the output message as a string (Unitless) d_{offset} is the distance between the target position and the landing position (m) p_{target} is the target position (m) ε is the hit tolerance (Unitless)
Notes	d_{offset} is from IM:offsetIM. The constraint $p_{\text{target}} > 0$ is from A:posXDirection. The constraint $d_{\text{offset}} > -p_{\text{target}}$ is from the fact that $p_{\text{land}} > 0$, from A:posXDirection. ε is defined in Sec:Values of Auxiliary Constants.
Source	–
RefBy	FR:Output-Values and FR:Calculate-Values

Preconditions for IM:messageIM

-

4.2.14 Data Constraints

The Data Constraints Table shows the data constraints on the input variables. The column for physical constraints gives the physical limitations on the range of values that can be taken by the variable. The uncertainty column provides an estimate of the confidence with which the physical quantities can be measured. This information would be part of the input if one were performing an uncertainty quantification exercise. The constraints are conservative to give the user of the model the flexibility to experiment with unusual situations. The column of typical values is intended to provide a feel for a common scenario.

Table 4: Input Data Constraints

Var	Physical Constraints	Typical Value	Uncert.
p_{target}	$p_{\text{target}} > 0$	1000 m	10%
v_{launch}	$v_{\text{launch}} > 0$	100 $\frac{\text{m}}{\text{s}}$	10%
θ	$0 < \theta < \frac{\pi}{2}$	$\frac{\pi}{4}$ rad	10%

4.2.15 Properties of a Correct Solution

The Data Constraints Table shows the data constraints on the output variables. The column for physical constraints gives the physical limitations on the range of values that can be taken by the variable.

Table 5: Output Data Constraints

Var	Physical Constraints
p_{land}	$p_{\text{land}} > 0$
d_{offset}	$d_{\text{offset}} > -p_{\text{target}}$
t_{flight}	$t_{\text{flight}} > 0$

5 Requirements

This section provides the functional requirements, the tasks and behaviours that the software is expected to complete, and the non-functional requirements, the qualities that the software is expected to exhibit.

5.1 Functional Requirements

This section provides the functional requirements, the tasks and behaviours that the software is expected to complete.

Input-Values: Input the values from Tab:ReqInputs.

Verify-Input-Values: Check the entered input values to ensure that they do not exceed the data constraints. If any of the input values are out of bounds, an error message is displayed and the calculations stop.

Calculate-Values: Calculate the following values: t_{flight} (from IM:calOfLandingTime), p_{land} (from IM:calOfLandingDist), d_{offset} (from IM:offsetIM), and s (from IM:messageIM).

Output-Values: Output t_{flight} (from IM:calOfLandingTime), s (from IM:messageIM), and d_{offset} (from IM:offsetIM).

Table 6: Required Inputs following FR:Input-Values

Symbol	**Description**	**Units**
p_{target}	Target position	m
v_{launch}	Launch speed	$\frac{\text{m}}{\text{s}}$
θ	Launch angle	rad

5.2 Non-Functional Requirements

This section provides the non-functional requirements, the qualities that the software is expected to exhibit.

Correct: The outputs of the code have the properties described in Properties of a Correct Solution.

Verifiable: The code is tested with complete verification and validation plan.

Understandable: The code is modularized with complete module guide and module interface specification.

Reusable: The code is modularized.

Maintainable: The traceability between requirements, assumptions, theoretical models, general definitions, data definitions, instance models, likely changes, unlikely changes, and modules is completely recorded in traceability matrices in the SRS and module guide.

Portable: The code is able to be run in different environments.

5.3 [Rationale —SS]

[Capture the rationale for the scope assumptions and final theory assumptions. The rationale could vary between problems. For instance, for projectile motion the rationale could be that it is being used for teaching purposes. If the theories are used to solve an actual science or engineering problem, the rationale would need more justification. —SS] [Neglecting rotation could be justified by assuming a point mass? A:pointMass) —SS]. [Should requirements be added related to guaranteeing assumptions and constraints? (As is done after a hazard analysis.) Requirements could be added to check the input constraints, like $x > 0$. Requirements could be added to check neglecting curvature. Would need the radius of the planet, or are we assuming it's Earth? —SS]

6 Traceability Matrices and Graphs

The purpose of the traceability matrices is to provide easy references on what has to be additionally modified if a certain component is changed. Every time a component is changed, the items in the column of that component that are marked with an "X" should be modified as well. Tab:TraceMatAvsA shows the dependencies of the assumptions on each other. Tab:TraceMatAvsAll shows the dependencies of the data definitions, theoretical models, general definitions, instance models, requirements, likely changes, and unlikely changes on the assumptions. Tab:TraceMatRefvsRef shows the dependencies of the data definitions, theoretical models, general definitions, and instance models on each other. Tab:TraceMatAllvsR shows the dependencies of the requirements and goal statements on the data definitions, theoretical models, general definitions, and instance models.

Table 7:

	A:twoD	**A:cartSyst**	**A:yAxisGravity**	**A:launchOrigin**	**A:targetXAxis**
A:twoD					
A:cartSyst					
A:yAxisGravity					
A:launchOrigin					
A:targetXAxis					
A:posXDirection					
A:constAccel					
A:accelXZero					
A:accelYGravity			X		
A:neglectDrag					
A:pointMass					
A:freeFlight					
A:neglectCurv					
A:timeStartZero					

Table 7: Trace

	A:twoD	A:cartSyst	A:yAxisGravity	A:launchOrigin	A:targetXAxis
A:gravAccelValue					

T

	A:twoD	A:cartSyst	A:yAxisGravity	A:launchOrigin	A:targetXA
DD:vecMag					
DD:speedIX					
DD:speedIY					
TM:acceleration					
TM:velocity					
GD:rectVel					
GD:rectPos					
GD:velVec	X	X			
GD:posVec	X	X			
IM:calOfLandingTime			X	X	X
IM:calOfLandingDist			X	X	
IM:offsetIM					
IM:messageIM					
FR:Input-Values					
FR:Verify-Input-Values					
FR:Calculate-Values					
FR:Output-Values					
NFR:Correct					
NFR:Verifiable					
NFR:Understandable					
NFR:Reusable					
NFR:Maintainable					
NFR:Portable					

Table 9: Traceabilit

	DD:vecMag	DD:speedIX	DD:speedIY	TM:acceleration	TM:velo
DD:vecMag					
DD:speedIX	X				
DD:speedIY	X				
TM:acceleration					
TM:velocity					
GD:rectVel				X	
GD:rectPos					X
GD:velVec					
GD:posVec					
IM:calOfLandingTime			X		
IM:calOfLandingDist		X			
IM:offsetIM					
IM:messageIM					

	DD:vecMag	DD:speedIX	DD:speedIY	TM:acceleration	TM:velo
GS:targetHit					
FR:Input-Values					
FR:Verify-Input-Values					
FR:Calculate-Values					
FR:Output-Values					
NFR:Correct					
NFR:Verifiable					
NFR:Understandable					
NFR:Reusable					
NFR:Maintainable					
NFR:Portable					

The purpose of the traceability graphs is also to provide easy references on what has to be additionally modified if a certain component is changed. The arrows in the graphs represent dependencies. The component at the tail of an arrow is depended on by the com-

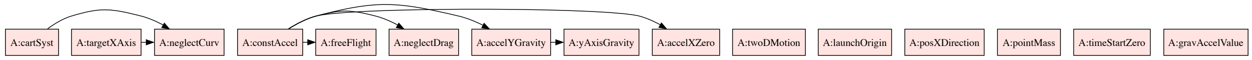

Figure 3: TraceGraphAvsA

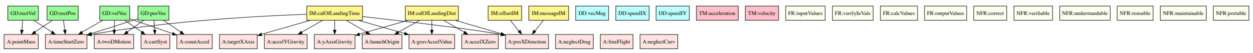

Figure 4: TraceGraphAvsAll

ponent at the head of that arrow. Therefore, if a component is changed, the components that it points to should also be changed. Fig:TraceGraphAvsA shows the dependencies of assumptions on each other. Fig:TraceGraphAvsAll shows the dependencies of data definitions, theoretical models, general definitions, instance models, requirements, likely changes, and unlikely changes on the assumptions. Fig:TraceGraphRefvsRef shows the dependencies of data definitions, theoretical models, general definitions, and instance models on each other. Fig:TraceGraphAllvsR shows the dependencies of requirements and goal statements on the data definitions, theoretical models, general definitions, and instance models. Fig:TraceGraphAllvsAll shows the dependencies of dependencies of assumptions, models, definitions, requirements, goals, and changes with each other.

For convenience, the following graphs can be found at the links below:

- TraceGraphAvsA
- TraceGraphAvsAll
- TraceGraphRefvsRef
- TraceGraphAllvsR
- TraceGraphAllvsAll

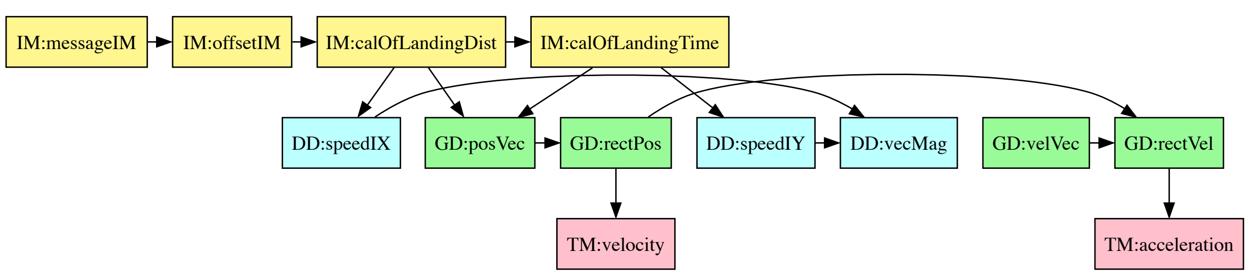

Figure 5: TraceGraphRefvsRef

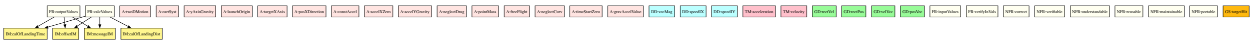

Figure 6: TraceGraphAllvsR

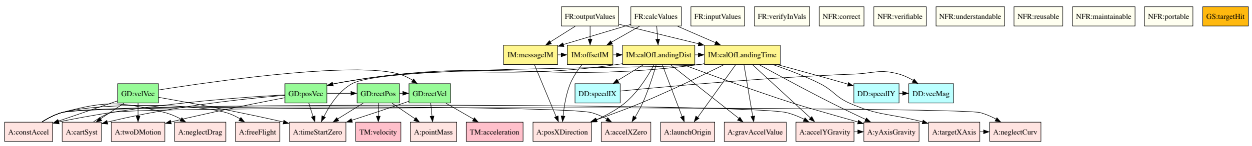

Figure 7: TraceGraphAllvsAll

7 Values of Auxiliary Constants

This section contains the standard values that are used for calculations in Projectile.

Table 11: Auxiliary Constants

Symbol	Description	Value	Unit
g	magnitude of gravitational acceleration	9.8	$\frac{\text{m}}{\text{s}^2}$
ε	hit tolerance	2.0%	–
π	ratio of circumference to diameter for any circle	3.14159265	–

8 References

[1] Wikipedia Contributors. *Acceleration*. https://en.wikipedia.org/wiki/Acceleration. June 2019.

[2] Wikipedia Contributors. *Cartesian coordinate system*. https://en.wikipedia.org/wiki/Cartesian_coordinate_system. June 2019.

[3] Wikipedia Contributors. *Velocity*. https://en.wikipedia.org/wiki/Velocity. June 2019.

[4] R. C. Hibbeler. *Engineering Mechanics: Dynamics*. Pearson Prentice Hall, 2004.