THE UNIVERSITY OF SYDNEY

ZERO ROBOTICS
ISS PROGRAMING CHALLENGE

Zero Robotics Australian Handbook

For participating students and teachers 2018

Faculty of Engineering and Information Technologies

THE UNIVERSITY OF SYDNEY

ZERO ROBOTICS
ISS PROGRAMING CHALLENGE

Contents

INTRODUCTION

PURPOSE OF THIS HANDBOOK

This handbook is designed to help students and teachers set up, manage and coordinate their efforts in participating in Zero Robotics and towards running their code on-board the International Space Station. The Zero Robotics website remains the main source of information about the competition. This handbook aims to shape expectations of the competition and provide practical advice and suggestions for Australian teams on how to best perform in the competition and fully benefit from the learning experience it offers.

HOW TO USE THIS HANDBOOK

If you are new to Zero Robotics we recommend you go through all the sections of the handbook in the sequence in which they are presented to get an overview of the competition, the types of challenges you will be presented with and recommended approaches to their solutions.

If you have participated in Zero Robotics before, feel free to browse through the handbook, look for sections that specifically interest you, or simply head to the FAQ section to find answers to questions that you may have had previously.

ABOUT ZERO ROBOTICS

Zero Robotics (created by the Massachusetts Institute of Technology and the University of Sydney's former NASA astronaut, Professor Gregory Chamitoff) is a robotics programming competition where the robots are SPHERES (Sychronised Position Hold, Engage and Reorient Experimental Satellites) that are aboard the International Space Station. The goal of the competition is to build critical engineering skills for students such as problem solving, design thought processes, operations training, and team work.

The competition is run through the Zero Robotics website (**http://zerorobotics.mit.edu**). Teams write and run their code on the website in order to control their SPHERES robot to solve an annual challenge.

The University of Sydney has partnered with the Massachusetts Institute of Technology (MIT) as regional coordinator for Australia. The University is supporting this program to encourage further uptake of STEM subjects and inspire students to explore the exciting range of STEM study and career opportunities.

THE COMPETITION

Every year MIT release a new game premise for the competition. Teams must figure out how to move their robot (including flying around and rotating) while executing specific tasks (e.g. picking up items, taking pictures of an opposing SPHERE). Points are

allocated based on the objectives of the game. The tasks become more complex through different stages of the competition.

LEARNING OUTCOMES

The competition is designed to give students experience in coding. Students will also learn the maths and physics behind the motion of the SPHERES robot and develop strategies for successful game play within the game premise. Students will need to work as a team, delegate tasks, communicate effectively and be well organised. These are valuable skills that students can apply to their schoolwork and that will be invaluable should they consider further studies post high school.

AUSTRALIAN QUALIFYING COMPETITION

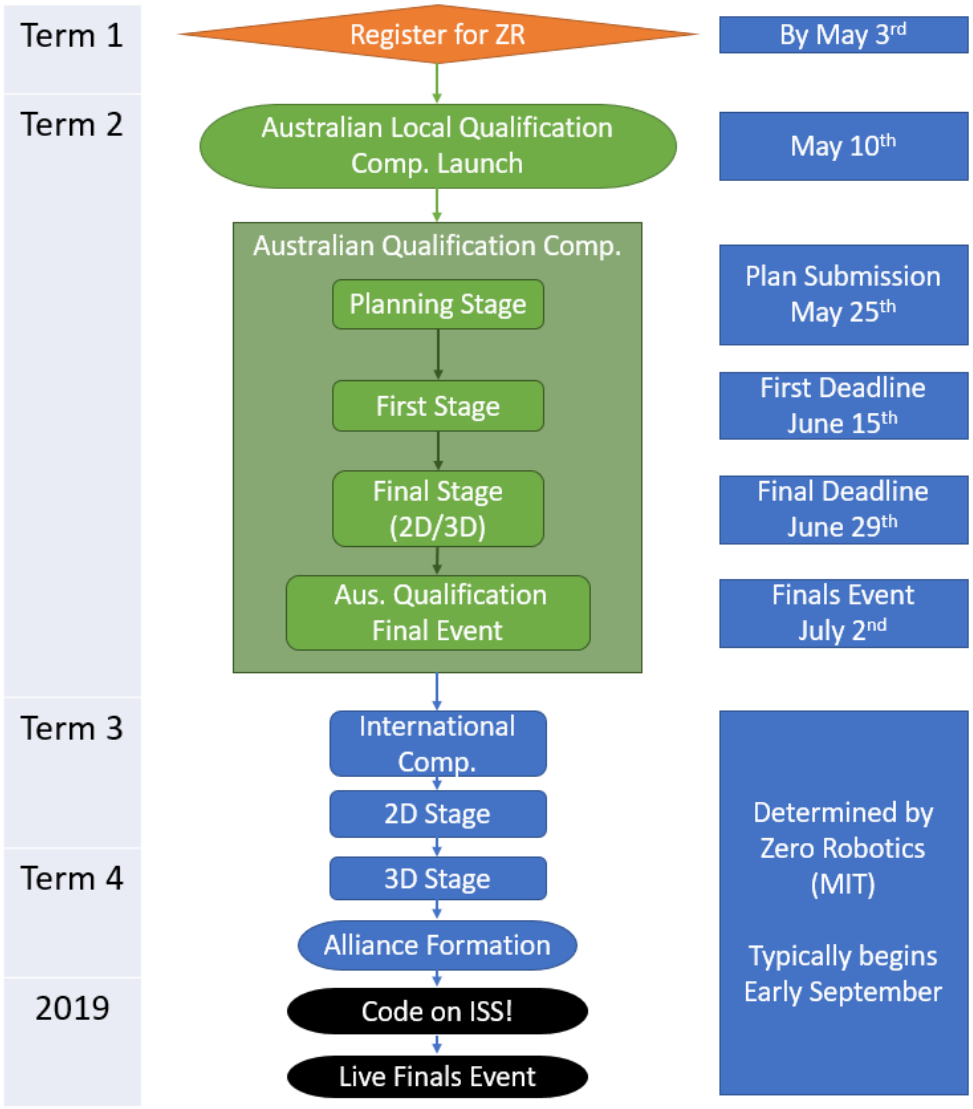
The Australian Qualifying competition allows students to get into some Zero Robotics programming before the international competition starts.

It is a great opportunity to get to know your team mates and figure out effective team roles and management strategies.

The International Zero Robotics Competition will see teams compete to make it to the finals, held on the International Space Station in mid-January.

DATES

Specific dates for the International competition will be released closer to the competition start.



GETTING STARTED

In this section you will find information on how to set-up your team, start working on the code and approach problems within the game.

CHOOSING STUDENTS FOR THE TEAM

Each school can be represented by a limited number of students. Zero Robotics rules stipulate that teams must be between 5-20 members. From our experience in past competitions, we suggest the ideal number of team members to be 8-10 students. Previous coding experience is not necessary and teams should not be selected based on this criteria as the team would benefit from a range of skill-sets. We suggest assembling a group of students whose skills are diverse. Some useful skills include:

- Maths
- Leadership
- People with good spatial imagination
- Physics
- Communication
- Strategic thinkers
- Coders
- Problem solvers

MEETING TIMES AND FREQUENCY

Once you've selected your team, it is important to organise regular meeting times. There are multiple, parallel ways that engagement in the competition can occur. Each team will need to establish a method that works best for them. An example meeting schedule could look something like:

- Team meetings 1-2 times per week (lunch/before/after school)
- Workshops with the teacher in charge – fortnightly
- Workshops/events at the University of Sydney

TUTORIALS TO GET YOU STARTED

If you are new to Zero Robotics

1) Get your SPHERES ready as described in "Make Paper SPHERES" tutorial on the Zero Robotics website:
http://static.zerorobotics.mit.edu/docs/ms/MakeYourOwnSPHERE.pdf

2) If your team is feeling confident and wants to get started have a look at the first few tutorials found under the 'Beginner' heading:
http://zerorobotics.mit.edu/tutorials

3) Video introductions and tutorials will be available throughout the year at the Australian Zero Robotics website.

For experienced coders and previous participants
Experienced coders may want to start having a look at how the game is played. We suggest having a look at the 2016 and 2017 games. Information and instructions can be found here: **http://zerorobotics.mit.edu/tournaments/24**

Previous participants might like to go through their strategy from 2015/2016 and problem solve issues that arose. We suggest going back though previous games (as above) and trying different strategies and troubleshooting how effective/ineffective different strategies could be.

DEVELOPING BASIC SKILLS

The Zero Robotics website has a range of tutorials to assist you through the learning process of coding and controlling the SPHERES. **http://zerorobotics.mit.edu/tutorials**

Tutorials are segmented into "Beginner", "Intermediate", and "Advanced". If you are relatively inexperienced in coding, we suggest working through sequentially. More experienced teams might like to find applicable tutorials as they see fit.

Make sure you look at the ZR User API in the Reference Document on the Tutorials page. This contains important information on how to talk to the SPHERE using the code.

We also suggest following community tutorials such as the y0b0tics! Circle Tutorial which can be found here:
http://www.yobotics.org/index.php?option=com_content&view=article&id=67&Itemid=270

*Tip: if you encounter any difficulties with using the functions described in the game API or game manual, refer to the Zero Robotics Forum (**http://zerorobotics.mit.edu/forum**). If there isn't an answer to your question, make sure you ask the teacher in charge or head to the Q&A section of the Zero Robotics website to post the question to us!*

TEAM COMPOSITION

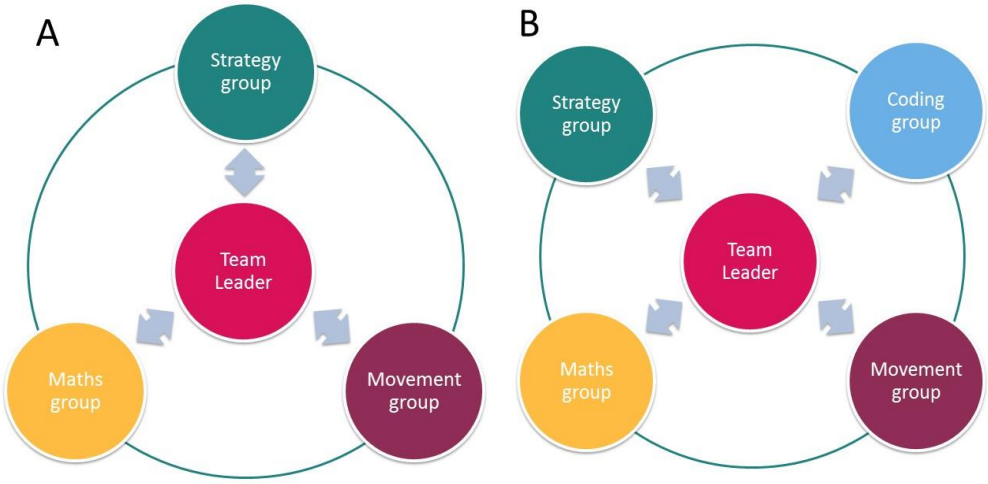
STRUCTURING THE TEAM

After trying out some of the activities advised in the previous section you might find team members naturally gravitate to particular tasks. This can be a good starting point when allocating roles within the team. Team dynamics will be different for each school but we have provided two examples below of how to structure your team.

The below diagram shows two examples with lines indicating communication channels and arrows showing assignment of tasks.

Structure A – no dedicated 'coding team'. Each team writes its own functions in code and the leader integrates and compiles the information.

Structure B – has a dedicated 'coding team'. They are responsible for encoding the maths and the strategy developed by other team members.



Once you have established your team's structure, you might want to divide tasks as per below:

Table 1. Example of Tasks divided between the groups.

Group	**Tasks**		
Strategy Group	Develop several **scenarios** in the game to account for.	Develop several game **strategies** for the SPHERE to follow.	Present a list of factors upon which the **decision** about taking an action is taken (e.g., pick up item if...).
Movement Group	List the **functions** of the SPHERE used in the game with their parameters.	Develop a function for **motion** of the SPHERE. I.e. how to get to where strategy team wants to go.	Develop function for execution of specific **actions** (e.g., pick up item, take photos, upload photos, etc.).
Maths Group	Provide the **parameters** and the **calculation** for the motion function of the SPHERES.	The **calculation** of distances, angles and relative position of the SPHERE.	Define **mathematical checks** for termination of each function.

Coding Group	**Establish conventions** for variable naming and code setup and maintain these.	**Integrate and translate** the plans/functions/code of the other groups into the team code.	**Debug** the code and manage the version control.

TEAM ROLES

As well as dividing your team into different groups, you might want to consider allocating specific roles to individual team members as highlighted in the example below:

Table 2. Example of specific roles within the team and associated tasks

Role	Tasks
Team leader	Distribute and divide tasks to other members. Ensure deadlines are communicated to team members and adhered to. Track progress and completion of specific tasks
Chief Coder	Check requirements for each function Integrate the team's code and be the master of the team's current code version. Regularly submit code to the ZR website.
Liaison officer/Team spokesperson	Be the main intermediary for communication between the team and the teacher in charge. Be the main intermediary for communication between the team and the University of Sydney.

INTERNAL COMPETITION

When you first start getting up to speed with the Zero Robotics competition it may be beneficial to split your team into two independent teams (or three depending on numbers). This can give everyone a chance to become familiar with different roles as well as test out different game strategies and coding methods. The internal teams can then compete against each other to find the best overall approach. See the tutorial from the Zero Robotics website on how to compete internally:
http://static.zerorobotics.mit.edu/docs/tutorials/IntramuralGameMode.pdf

WORKLOAD PLANNING

It is important to become very familiar with the schedule of the competition and plan your workload around other competing commitments at school. Zero Robotics is designed to complement your school work so it's important you manage your time effectively around your homework and other extracurricular activities.

During the competition you may find that some deadlines fall outside of term time (particularly if you move through to the international competition). Always plan for early submission. This will alleviate any issues with communicating with your team during school holidays or busier times (e.g. when there's a school camp or assessment) at school.

Plan conservatively and take into account commitments of ALL team members when estimating how much your team can achieve from week-to-week.

CODING AS A TEAM

When multiple members of the team have access to the same source code, tracking modifications can become tricky. It's a good idea for the team to agree on procedures to ensure effective collaboration without losing important information and mixing up the source files. ALWAYS have a back-up plan with your code – i.e. a safe, operational code for submission if your more complex code can't be completed in time. If you have time to complete the complex version – great! If not, you still have something you can submit.

An example procedure can be:

- Give the Chief Coder final control over the current, most up-to-date version of the code (master code). All changes to the master code need to go through him/her and they are responsible for code submission.
- Clearly name each document prior to saving – come up with a convention for your team (e.g. filename_{name of person editing}_date, *testcode_Ben_04_06*)
- Save REGULARLY
- Be careful to distinguish between the master code and additional functions under development to keep one stable working version.

Tip: Limiting the number of people editing the master code (main working code) of your program and keeping a team log of changes from the start can become very handy during later stages of the competition and help you avoid unnecessary confusion.

COMMUNICATION

COMMUNICATION METHOD

Communication between the Zero Robotics competitors is crucial to successful collaboration. Our preferred method of communication is Slack.

Slack is a cloud based team collaboration software that allows users to instantly message each other across a variety of channels (topics). It has seen widespread uptake in engineering and project management fields, and is best suited to small group projects like Zero Robotics.

Slack supports code snippets and equation formatting much more easily than other communication methods, allowing questions and answers to be clear and easy to understand.

Generally each topic has a dedicated channel which participants can subscribe to; for example we might use the channels #coding_practice, #strategy, #rules_questions and #prelim_comp for a start and expand as the program progresses.

Please let us know if you would like more information or have questions about the communication method.

The team Liaison Officer takes a leading role in communication and is responsible for ensuring all of the team, and teachers, are up-to-date with the team's progress.

COMMUNICATION DURING INTERNATIONAL ALLIANCE ROUND

In the later stages of the competition, successful teams form international alliances comprising of three teams. This part of the competition can present challenges as you will likely be communicating across multiple international time zones with people you've never met (face-to-face). During this part of the competition, your priorities should be establishing communication methods, communication times, collaboration practices and roles and responsibilities across teams.

ROLE OF THE TEACHER IN CHARGE

The teacher's role is to provide guidance and advice to the team. It is NOT a requirement for the teacher to have coding experience. This support can be provided through the Q&A sessions and the available online tutorials. It is, however, strongly advised, that the teacher is involved closely through the entire competition. Examples of tasks the teacher may be required to fulfil:

- Ensure the team sets up weekly meetings
- Ensure all members of the team are active participants and have roles
- Motivate the team to continue through the competition.

- Identify any potential barriers to the team's performance and address these issues either internally.
- Ensure the team access materials provided
- Ensure the team have enough technical support and coordinate and deliver workshops as required.
- Liaise with the Zero Robotics Australian Coordinator at the University of Sydney if any issues arise.
- Provide feedback to the Zero Robotics Australian Coordinator regarding the organisation of the competition.

FAQs

In this section you will find answers to some frequently asked questions. For more detailed discussion on topics that interest you, refer to the forum section on the Zero Robotics website: **http://zerorobotics.mit.edu/forum**.

Q: The SPHERE isn't doing what we expect. What's going on? How do the SPHERES move around?
A: The simulations run for Zero Robotics are using a research grade simulator and hence are modelled off the physics of the real SPHERES robot with associated physical limitations. The SPHERES move around with 12 CO_2 gas thrusters. These thrusters release pulses of compressed CO_2 to translate and rotate the SPHERE.
It is up to the teams to investigate how that effects their performance.
Your best resource is the information on the SPHERES website:
http://ssl.mit.edu/spheres/index.html

Q: We are having troubles with our submission, can you help?
A: To minimise problems with code submission, aim to submit in advance of the deadline (at least one or two days prior). This will allow you time in case the server is not responding. The server often undergoes maintenance or is busy performing calculations during US night-time. If this is the case, organisers will let participants know about scheduled maintenance.

Q: Why am I having issues with my code size being too large?
A: There is a restricted amount of memory and processing power on the real SPHERES robots on the International Space Station, hence the Zero Robotics code has to a limited size. Use the "Codesize Esimate" tool available under "Simulate" when you are at the project page to see how large your code is. There are many factors that affect the code size including: the number of lines of code, the number and types of variables (floats take more memory than int's which take more memory than bool's), and complexity of your code (number of loops and team functions).

Q: What maths do we need to know for Zero Robotics?
A: Zero Robotics is a great opportunity to learn about mathematics concepts applied to a specific problem: controlling robots in space! Some of the key maths concepts used

are: vectors and vector operations (dot and cross products) to work with objects positions, orientations and relative positions, polar coordinates, trigonometry, circular motion, and working with equations of motion to predict and control the robots. General maths based problem solving is also used a lot in check conditions for code and the logic of programming. It is useful to know these concepts, but they can also be learned through the competition.

Q: I don't know anything about coding. Where do I start?
A: Zero Robotics is a great way to learn! Starting with the very basics, it can be useful to try graphical, block programming. Scratch has some great introductions to do this (**https://scratch.mit.edu/**), and the hour of code (**https://code.org/learn**) can also be a good starting point. You can also use block programming for Zero Robotics. You can start the basics of controlling the SPHERES with tutorials explained using the "Graphical Editor", which is very similar to Scratch (**http://zerorobotics.mit.edu/ms/tutorials/**). You can then progress to trying C code with the tutorials for the "Text Editor" (**http://zerorobotics.mit.edu/tutorials/**).

Q: I love Zero Robotics! How can I get involved in more activities like this? A: The University of Sydney has a range of programs, clubs, and workshops that you can get involved in. Visit: **http://sydney.edu.au/engineering/high-school**

Q: Where can programming and robotics take me at university and into my career?
A: With technology becoming more and more prevalent in our lives, the ability to program and work with technology is hugely important in a wide range of fields, from healthcare, to finance, to journalism to engineering and IT. The types of degrees at university that can help you learn about robotics are primarily in the engineering and IT fields. These include mechatronics engineers (working with robots), aeronautical engineering (aircraft and spacecraft), space engineering (working with satellites, space robots, rockets), and IT (using coding to create a wide range of useful software).

CONTACTS

Zero Robotics Coordinator:

Penny Player
Faculty of Engineering and Information Technologies
University of Sydney
penelope.player@sydney.edu.au

Zero Robotics Coordinator:
Jordan Fazio-Nagy
Faculty of Engineering and Information Technologies
University of Sydney
jordan.fazio-nagy@sydney.edu.au

Zero Robotics Coordinator:
Felix Best
Faculty of Engineering and Information Technologies
University of Sydney
felix.best@sydney.edu.au

APPENDIX

The table below outlines what can be learned in each of the tutorials on the Zero Robotics Website (http://zerorobotics.mit.edu/tutorials). Refer to this appendix to find tutorials to help with questions you have, or to see what you can learn more about. The tutorials also include off computer activities to introduce the basics of programming and learn more about the SPHERES and physics concepts.

Tutorial title	**Content and tips**
WEBSITE INTERFACE	
Create an Account, Invite Students to Your Team	How to set up account in the website (needs a Google account), and invite students to a team.
Getting to know ZR IDE	How to start a new project, compile and simulate your code.
Using the Graphical Editor	Has a good walkthroughs to describe the basics of C++ programming (variable types, naming conventions etc.). Details on using the code block based graphical editor (like scratch).

Intro to Game Mode	Creating code for a specific game scenario. How to select a standard opponent.
Intramural Game Mode	How to share projects with your team. How to set up a match between different sets of code. How to **Submit Code**. *The process required to submit the final code for competition*
BEGINNER. Basic SPHERES Controls, Programming Concepts, Useful Math and Physics	
Basic SPHERES Controls	
Variables, Arrays, and the **setPositionTarget** Function	Intro to Variables, Arrays and the **setPositionTarget** Function. How to use **API functions.**
More Simple Arrays and **setAttitudeTarget**	Basic intro of how SPHERES move and rotate. Description of coordinate axes. How rotation is defined (giving a pointing vector for the –x face). Example of commanding an attitude (where the SPHERE is pointing) change.

Programming Concepts	
Conditionals: The basics of "If-Then"	Intro to "if" statements – flow diagrams and how to code in C. Simple examples to learn about "if" statements. Logical > operator, ++ to step a variable.
Conditionals: More Fun with "If-Then", Logic Operators and Debugging	What the program does each second on the robot. More examples with logical operators. Debug statements (and where they show).
Conditionals: Advanced Logic Operators	"AND" and "OR" operators – introduction and examples.

Conditionals: "Else"-"If"	Introduction to "else" and "else-if" and examples to try it out.
"For" Loops	Introduces two new API functions (**getMyZRState** and **getOtherZRState**). Computing *mid points*. First use of *local variables*. Example using a "for" loop to compute a mid-point (one loop per dimension).
Applied Conditionals	More on **getMyZRState** and what the output is. Using the position info for controlling the SPHERE. *Small note on compensating for SPHERE control system realities (inaccuracies).*
Creating Functions	Introduction of and creation of a simple function.
Functions and the Step Counter Model	Explains limitations of previous projects for doing multiple tasks, especially as the loop is called once per second. Introduction of a step counter to split code into different steps. Then implements a step counter with different functions to move the SPHERE to a target. *Good base to understand how to start a game code.*
Useful Maths and Physics	
Intro to Physics	Weight and mass. Rigid bodies. Translation and rotation.
Kinematics	Displacement, velocity, acceleration and momentum.
	Vectors vs. scalars. Equations of motion.

Combining Vectors	Basics on vectors & vector arithmetic (+, -, magnitude). Code for working with vectors, creating vectors. Referencing terms in vectors for C. **mat_matrix.h** functions – mathVecSubtract() etc.
INTERMEDIATE. More SPHERES Controls	
setAttitudeTarget, Revisited	Unit vectors. Using vector arithmetic to keep the SPHERE attitude pointing toward a target while moving.
Hints about SPHERES Loop Dynamics	Physics of SPHERES Dynamics (how thrusters are used). Newton's First Law. Some trials with coding to see how the dynamics affects how the code will control the SPHERE. Drifting translation/rotation if code stops commanding the SPHERE. *Important understanding for programming for the game.*
setVelocityTarget	How to use **set Velocity Target**. Ideas to use this to make manoeuvres more quickly. Discussions on linear momentum. Some points on for loop syntax. *First tutorial with no walkthrough.*
Force	**setForces** function and how it is used. Open Loop control.
Spinning: Torque	**setTorque.** Introduce body and global frame. Intro to torque, formula. Signs (=/-, Right Hand Rule).
Spinning: Angular Velocity	Basics on angular velocity – angular velocity vector. All units in radians. **setAttRateTarget.** Angular momentum discussion (with Inertia).

Spinning: Partial Turns	Angular displacement and dot products. **mathVecInner** – for inner dot product, acosf(). Approach to command a specific angular displacement.

	Equations of motion to help designing a good strategy.
ADVANCED	
Spinning: tilted Axis	Using cross product to rotate around a tilted axis. How to compute the cross product with matrices **mathVecCross.** Set attitude target as cross product of attitude and rotation vector. *Only a starting point – invite students to be creative.*
Revolving: Polar Coordinates	Introduction to polar coordinates. Uniform circular motion. Using polar coordinates to command uniform circular motion (with position and attitude targets). *Setting Waypoints – suggestions.*
Revolving: Tilted Axis	Revolving around a general centre and rotation axis using the cross product to identify a plane of rotation. Open challenge to command an "orbit" efficiently around an arbitrary axis, and to maintain pointing towards the object. *Important capability to develop for many games.*
Recommended Functions	General advice on writing functions. List of functions that are useful. Functions to get different info on the current states and relations of SPHERE, other SPHERE, targets, position and dynamics. Functions to control efficient motion (fast and stable motion). ***moveFast, spin, revolve.*** Collision avoidance. *Some functions specific to the game.*
OPTIONAL TEAM ACTIVITIES SPHERES, Physics, Programming, Review	

SPHERES	
SPHERES	Make Paper SPHERES. Paper SPHERES Plan. Label the SPHERES.
Communications	Activity to understand SPHERES communication operation.

Trilateration Ropes	Activity in classroom with ropes to understand how SPHERES positions are computed.
Physics	
Grids and Coordinates	Clues and treasure hunt with questions using coordinates and grids and with clues laid out with grid coordinates.
Vector Hunt	Describing vectors with orientation and direction – get a physical sense for vectors.
Soccer Ball Demo	Activity to understand different aspects of Newton's first law.
Chairs on Wheels	Newton's Third Law – equal and opposite reaction on wheeled chairs.
Thruster Balloons	Explaining Balloon movement with Newton's third law.
Free Body Diagrams	Example problems to draw free-body diagrams (and solutions).
Bottle Rockets	Activity to explain kinematics, Newton's Laws. Students design their own bottle rockets.
Bouncy Balls	Use a bouncy ball to discuss forces and dynamics.

Dry Ice Dynamics	Using dry ice to visualise dynamics in a near frictionless environment (such as for the SPHERES).
Speed of Sound	Study difference between speed of sound and speed of light.
Programming	
Programming Peanut Butter and Jelly	Activity to explain type of thorough descriptions required for programming. Idea that the computer takes everything literally. Get in mindset for debugging code.
Write it then Do it – Programming	Reinforce thoroughness required in programming. Live activity of the debugging process.
Review	
Jeopardy	Quiz with 5 questions on each of Space / Programming and logic / Maths and Physics / Zero-G IDE / SPHERES.
Team Management FAQ	Tips to setting up and organising the personnel in a team. Tips on how to run the team and approach to developing the code/strategies/functions.

THE UNIVERSITY OF SYDNEY

ZERO ROBOTICS

ISS PROGRAMING CHALLENGE