Software Engineering

Stage 6 (Year 11) – teacher support resource

Programming mechatronics

# Teacher support resource

**Teacher note**: this resource has been designed to facilitate the ready conversion into a student booklet by removing the answers within the response windows. Teacher notes can be deleted before distributing to students.

Student name:

Class:

Teacher:

Contents

[Teacher support resource 1](#_Toc159418411)

[Assessment task overview 7](#_Toc159418412)

[Steps to success 8](#_Toc159418413)

[What is the teacher looking for? 8](#_Toc159418414)

[Glossary 11](#_Toc159418415)

[NESA glossary keywords 14](#_Toc159418416)

[Understanding mechatronic hardware and software 15](#_Toc159418417)

[Outline applications of mechatronic systems in a variety of specialised fields 15](#_Toc159418418)

[Identify the hardware requirements to run a program and the effect on code development 17](#_Toc159418419)

[Assessing the relationship of microcontrollers and the central processing unit (CPU) 19](#_Toc159418420)

[The influence of instruction set and opcodes 21](#_Toc159418421)

[The use of address and data registers 26](#_Toc159418422)

[Identify and describe a range of sensors, actuators, and end effectors/manipulators within existing mechatronic systems 29](#_Toc159418423)

[Inputs – transducers and sensors 32](#_Toc159418424)

[Motion sensors 32](#_Toc159418425)

[Outputs – actuators and end effectors 45](#_Toc159418426)

[Hydraulic actuators 48](#_Toc159418427)

[Robotic grippers 49](#_Toc159418428)

[Use different types of data and understand how it is obtained and processed in a mechatronic system, including diagnostic data and data used for optimisation 52](#_Toc159418429)

[Analog sensor data 54](#_Toc159418430)

[On/off digital sensor data 57](#_Toc159418431)

[Digital I2C sensor data 61](#_Toc159418432)

[Combined sensor data 63](#_Toc159418433)

[‘Analog’ outputs – pulse width modulation 63](#_Toc159418434)

[LED 64](#_Toc159418435)

[Motor 66](#_Toc159418436)

[Servo 67](#_Toc159418437)

[Digital outputs 72](#_Toc159418438)

[I2C outputs 74](#_Toc159418439)

[Experiment with software to control interactions and dependencies within mechatronic systems 77](#_Toc159418440)

[Motion constraints 77](#_Toc159418441)

[Degrees of freedom 82](#_Toc159418442)

[Combination of subsystems 84](#_Toc159418443)

[Combination of sensors, actuators and end effectors to create viable subsystems 90](#_Toc159418444)

[Determine power, battery and material requirements for components of a mechatronic system 95](#_Toc159418445)

[Develop wiring diagrams for a mechatronic system, considering data and power supply requirements 97](#_Toc159418446)

[Determine specialist requirements that influence the design and functions of mechatronic systems designed for people with disability 101](#_Toc159418447)

[Designing control algorithms 102](#_Toc159418448)

[Develop, modify and apply algorithms to control a mechatronic system 104](#_Toc159418449)

[Explore the algorithmic patterns, code and applications for open and closed control systems 109](#_Toc159418450)

[Outline the features of an algorithm and program code used for autonomous control 111](#_Toc159418451)

[Programming and building – user-programmable articulating arm 114](#_Toc159418452)

[Design, develop and produce a mechatronic system for a real-world problem 116](#_Toc159418453)

[Software control 116](#_Toc159418454)

[Gripper control 117](#_Toc159418455)

[Potentiometer control of servos 117](#_Toc159418456)

[Mechanical engineering 118](#_Toc159418457)

[Electronics and mathematics 119](#_Toc159418458)

[Connect and test buttons 121](#_Toc159418459)

[Implement algorithms and design programming code to drive mechatronic devices 125](#_Toc159418460)

[Develop simulations and prototypes of a potential mechatronic system to test programming code 128](#_Toc159418461)

[Apply programming code to integrate sensors, actuators and end effectors/manipulators 129](#_Toc159418462)

[Implement specific control algorithms that enhance the performance of a mechatronic system 135](#_Toc159418463)

[Design, develop and implement a user interface (UI) to control a mechatronic system 141](#_Toc159418464)

[Create and use unit tests to determine the effectiveness and repeatability of each component’s control algorithm 151](#_Toc159418465)

[Unit testing – samples 152](#_Toc159418466)

[Programming and building – self-stabilising ball on moving beam 155](#_Toc159418467)

[Design, develop and produce a mechatronic system for a real-world problem 156](#_Toc159418468)

[Software control 156](#_Toc159418469)

[Mechanical engineering 157](#_Toc159418470)

[Electronics and mathematics 158](#_Toc159418471)

[Implement algorithms and design programming code to drive mechatronic devices 164](#_Toc159418472)

[Develop simulations and prototypes of a potential mechatronic system to test programming code 165](#_Toc159418473)

[Design, develop and implement programming code for a closed loop control system 165](#_Toc159418474)

[Apply programming code to integrate sensors, actuators and end effectors/manipulators 167](#_Toc159418475)

[Implement specific control algorithms that enhance the performance of a mechatronic system 168](#_Toc159418476)

[Design, develop and implement a user interface (UI) to control a mechatronic system 169](#_Toc159418477)

[Appendix 181](#_Toc159418478)

[Equipment selection information 181](#_Toc159418479)

[Microcontrollers vs microcontroller boards 181](#_Toc159418480)

[Microcontroller board considerations 181](#_Toc159418481)

[Comments on specific microcontroller boards 182](#_Toc159418482)

[Sensors, actuators and other components 184](#_Toc159418483)

[Sensors and inputs 184](#_Toc159418484)

[Actuators, end effectors and outputs 184](#_Toc159418485)

[Supporting components 185](#_Toc159418486)

[Software considerations 185](#_Toc159418487)

[References 187](#_Toc159418488)

## Glossary

Many of the following words will gather more meaning to you as you work through this booklet.

Each time you see an unfamiliar word in bold throughout this workbook you can add its definition in the table below in case you need to refer to it later.

|  |  |
| --- | --- |
| Word | Definition |
| breadboard | A generic prototyping board that allows students to plug wires into conductive channels to make connections rather than soldering. |
| breakout board | A circuit board with microchip and other electronics to give easy access to pins and functionality of the microchip with standard pins or sockets on jumper wires.  These boards are excellent for prototyping and to avoid the need for soldering when experimenting with sensors and actuators in software engineering. |
| Grove | A proprietary wiring socket system that makes physical connections simpler and safer.  These generally require a grove hat, shield or adapter to use and may be simplest if all sensors and actuators use the Grove connectors. |
| hat, shield | An electronic board designed to fit onto the pins of a specific microcontroller board, breaking out some of its functionality in some way.  Some direct pin access functionality can be lost. |
| headers | A row of electrical pins that can be soldered to a circuit board to help make connections to breadboards or wires.  [Stackable headers](https://www.youtube.com/shorts/pR0w9-BBn1M) include an extended socket part to allow both plug and socket connections.  These can be particularly useful when using hats or shields that might otherwise limit pin availability. |
| horn, servo horn | A connector for the rotational shaft of a servo.  Enables the connection of rotational motion to devices. |
| light emitting diode (LED) | A semiconductor device that emits light when current flows through it. |
| motor controller | A motor driver with additional logic to ensure that attached motors are driven correctly.  These reduce chances of errors and mistakes by the programmer. |
| motor driver | An electronic chip or circuit board that uses a combination of input values to set the status of different power transistors to hopefully drive a connected motor.  The programmer needs to be careful to set the correct combination of input values to avoid damaging the circuit or motor. |
| potentiometer, POT | A potential divider circuit using a dial or slide to control its internal variable resistances. |
| servo | An actuator with angle or speed control.  Typically controlled using a pulse of 1–2ms in a >=20ms time window (that is, <=50Hz signal).  The width of the pulse specifies the desired angle or speed of the servo. |
| microcontroller board | A circuit board containing a microcontroller chip designed for prototyping mechatronic systems.  Programs can be uploaded to them for testing.  **Note:** some people use the term microcontroller to refer to microcontroller boards. |
| pinout | A specification stating the functionality of each pin of a microcontroller or microcontroller board.  Often shown as a labelled diagram. |

## NESA glossary keywords

|  |  |
| --- | --- |
| Word | Definition |
| Identify | Recognise and name |
| Describe | Provide characteristics and features |
| Explain | Relate cause and effect; make the relationships between things evident; provide why and/or how |
| Investigate | Plan, inquire into and draw conclusions about |
| Compare | Show how things are similar or different |
| Distinguish | Recognise or note/indicate as being distinct or different from; to note differences between |
| Recommend | Provide reasons in favour |

[NESA A Glossary of Key Words](https://educationstandards.nsw.edu.au/wps/portal/nesa/11-12/hsc/hsc-student-guide/glossary-keywords)

**Teacher note:** for students with an EAL/D background, the glossary and NESA keywords can be provided complete so that they have additional time to understand the key terms with bilingual dictionaries. The glossary can be provided to students in their preferred communication mode.

# Understanding mechatronic hardware and software

## Outline applications of mechatronic systems in a variety of specialised fields

**Activity 1**: mechatronic systems

****As a class, watch [Mechatronics (2:45).](https://www.linkedin.com/learning/top-10-skills-for-robotics-engineers/mechatronics) Discuss applications of mechatronic systems and complete the question and table below.

1. Describe mechatronics and explain its purpose.

|  |
| --- |
| Sample answer:  Mechatronics is the practical application of systems that combine software with electronics and mechanics to perform actions.  Mechatronic systems can perform tasks repeatedly, accurately, quickly and cheaply, so can make cost effective replacements for human labour.  They use sensors for input from the environment, actuators to create output and software to process the inputs and control the outputs. |

1. Complete the table on the next page to show examples of mechatronic systems.

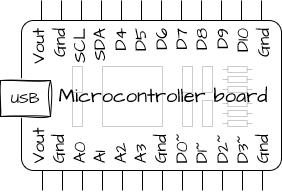
|  |  |  |
| --- | --- | --- |
| Mechatronics system | Description of mechatronic system | Image of mechatronic system |
| Home mechatronics system |  |  |
| Industrial mechatronics system |  |  |
| Medical mechatronics system |  |  |
| Transportation mechatronics system |  |  |

Identify the hardware requirements to run a program and the effect on code development

Mechatronic systems are designed to perform specific fixed tasks and, to save cost and space, they do not execute their code on a normal computer or powerful microprocessor (CPU). Mechatronic systems execute their code in a microcontroller which contains a very small microprocessor, some memory to store program code and data and input/output pins to sense and react to their environment.

To help prototype mechatronic systems, microcontrollers are widely available on circuit boards which include a method to connect and upload code. These boards are called microcontroller boards and may include extra functionality such as sensors, LEDs and buzzers.

Figure 1 – microcontroller board



**Activity 2**: microcontroller boards

1. Find and paste images of different microcontroller boards into the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| Arduino Uno | Arduino Nano | Raspberry Pi Pico | Micro:Bit |
|  |  |  |  |

Research the hardware and methods available to program a microcontroller board available in your school and complete the questions and tables below.

1. Describe the steps required to connect your microcontroller board to a computer and upload a program.

|  |
| --- |
| Sample answers could include:  The Pico can be plugged in via a USB cable and presents as a mass storage device. Compiled code from a language like C++ can be copied directly to it from the file browser. Micropython programs can be executed on the Pico if it has been configured with the right firmware through an IDE like Thonny.  The Arduino can be plugged in via a USB cable and presents as a mass storage device. Compiled code from a language like C++ can be copied directly to it from the Arduino IDE. |

1. Complete the table below for your microcontroller board.

|  |  |
| --- | --- |
|  | Microcontroller board model |
| Number of pins |  |
| Number of general-purpose input/output pins (GPIO) |  |
| Microcontroller model |  |
| Programmable memory available |  |
| Communications support during program execution (for example Wi-Fi, Bluetooth, radio, serial cable, USB) |  |
| Image of microcontroller board |  |

**Teacher note**: the table below could be used as an opportunity to identify compatibility issues connecting microcontrollers to computers or laptops used in your classroom.

1. Complete the table below.

|  |  |  |
| --- | --- | --- |
|  | Your answer | Will this work from your computer? |
| How is power provided to the microcontroller board? |  |  |
| Name or link to IDE or browser-based coding site for your board |  |  |
| Programming languages supported by the board’s microprocessor |  |  |
| Program upload method – Wi-Fi, Bluetooth, USB cable, etc |  |  |

### Assessing the relationship of microcontrollers and the central processing unit (CPU)

**Activity 3**: comparing microprocessors and microcontrollers

Read about the difference between [microprocessors and microcontrollers](https://hardwarebee.com/microcontroller-vs-microprocessor-what-to-choose/) to answer the question below.

1. Compare and contrast microprocessors and microcontrollers. In your response, refer to the typical memory, power consumption, size and/or price for each type of device.

|  |
| --- |
| Sample answer:  A microprocessor is a generic computing unit designed to perform calculations or logical operations, like a CPU. A microcontroller is a chip that includes a microprocessor as well as supporting hardware such as memory, analog to digital converters for sampling and pins for reading and writing data between the chip and external devices.  A microcontroller is bigger and more expensive than the microprocessor that it contains, but a complete system created with a microcontroller will be smaller and cheaper because the microcontroller already contains so many of the necessary extra parts. |

1. For your microcontroller board, research the microcontroller chip used and complete the table below. The following links may help:

* [ATmega328](https://en.wikipedia.org/wiki/ATmega328) (used in many Arduinos)
* [RP2040](https://en.wikipedia.org/wiki/RP2040) (used in the Raspberry Pi Pico)
* [nRF52833](https://www.nordicsemi.com/Products/nRF52833) (used in the Micro:Bit)

|  |  |
| --- | --- |
|  | Information for your microcontroller |
| Programming memory available (SRAM, RAM) |  |
| Microprocessor model (CPU) |  |
| Microprocessor speed (MHz) |  |
| Number of general-purpose IO pins (GPIO) |  |

1. **Challenge question:** Recommend and justify an appropriate microcontroller to use for a task that requires the following:

* digital data acquisition from 10 sensors
* high accuracy analog to digital sampling from 2 sensors
* complex processing of all sensor data
* digital control of 10 different devices
* non-volatile data logging of the system’s operations to keep information when the system loses power.

### The influence of instruction set and opcodes

The set of instructions a particular microprocessor can execute is called its instruction set. Instruction sets consist of binary codes but are usually described for human readers in a text-based assembly language. Binary instructions are given to the microprocessor with an operation code to represent the type of instruction followed by any parameters required by that instruction. An instruction, shown in assembly language, could look something like this:

ADD r1, r2   
In most common instruction sets, this instruction would add the contents of register1 and register2, storing the result back into register1.

**Activity 4**: programming a microprocessor

As a class, brainstorm reasons why microprocessors used by microcontrollers in mechatronic systems might need special instructions in their instruction sets. Consider the following typical mechatronic system design issues:

* the need to read sensors and control actuators
* the desire to conserve as much power as possible for longer runtimes on batteries
* the need to communicate with devices that use complex communication protocols.

1. Complete the table below.

|  |  |
| --- | --- |
| Consideration | Potential impact on instructions available or instruction set design |
| Reading sensors and controlling actuators | **Sample answer**: Mechatronic systems rely on various peripherals such as sensors and actuators Special instructions are tailored to efficiently interface with these peripherals to streamline data processing. |
| Power conservation | **Sample answer**: Mechatronic systems often operate in power-constrained environments away from mains supply. Special instructions can be designed to perform specific tasks with minimal power consumption, contributing to overall energy efficiency. |
| Complex communication protocols | **Sample answer**: Mechatronic systems rely on various communication interfaces. Special instructions are tailored to efficiently interface with peripherals to streamline communication. |

Read [Microprocessor Programming](https://www.allaboutcircuits.com/textbook/digital/chpt-16/microprocessor-programming/) and answer the following questions.

1. Describe how machine code and assembly language differ.

|  |
| --- |
| Sample answer:  Machine code is programming code executed in a processor. It is recorded in binary as that is the only form that a processor understands. Each machine code instruction consists of an opcode and operands. The opcode is the binary code for the instruction to perform (for example to add or multiply) and the operands are the data items used in the operation (for example values or memory addresses). Assembly language is a text-based abstraction of machine code to make reading the code easier for humans. An example of an assembly language instruction is MOV A, E. |

1. Explain the need for a translator when programming mechatronics systems.

|  |
| --- |
| Sample answer:  A translator changes high level source code such as C++ or Python into binary machine code for the embedded microprocessor in the microcontroller. Compiled languages have their source code translated in binary machine code all at once. This would typically happen on a developer’s computer before being copied to a microcontroller. Interpreted languages have their source code translated line by line at run time. For microcontrollers, this would mean that the interpreter would have to be running on the device. |

Use the sample assembly language program shown below. It is written for an AVR microprocessor (used in Arduino microcontroller boards) and is shown in assembly language.

LDI R16, 1

loop: OUT 1, R16

ADD R16, R16

BRVC start

1. Show the format of an instruction from an instruction set by completing the table below.

|  |  |  |
| --- | --- | --- |
|  | First part of instruction | Remainder of instruction |
| Instruction format |  |  |
| Instruction example |  |  |

1. Use the [AVR (Arduino) Instruction Set Manual](https://ww1.microchip.com/downloads/en/DeviceDoc/AVR-Instruction-Set-Manual-DS40002198A.pdf) (pp 23–146) to complete the table below.

Describe the purpose of each given line of the assembly language program shown.

The page number for each opcode instruction is listed for you.

|  |  |  |
| --- | --- | --- |
| Assembly instruction | AVR instruction set page reference | Description (Some have been completed for you already) |
| LDI R16, 1 | [p90](https://ww1.microchip.com/downloads/en/DeviceDoc/AVR-Instruction-Set-Manual-DS40002198A.pdf#_OPENTOPIC_TOC_PROCESSING_d138e44224) |  |
| loop: | [NA] | This is not an instruction – it is a label used so that branch instructions know where to go |
| OUT 1, R16 | [p105](https://ww1.microchip.com/downloads/en/DeviceDoc/AVR-Instruction-Set-Manual-DS40002198A.pdf#_OPENTOPIC_TOC_PROCESSING_d138e51118) | Copies the value stored in R16 to a device attached to output pin #1 |
| ADD R16, R16 | [p24](https://ww1.microchip.com/downloads/en/DeviceDoc/AVR-Instruction-Set-Manual-DS40002198A.pdf#_OPENTOPIC_TOC_PROCESSING_d138e15256) |  |
| BRVC start | [p49](https://ww1.microchip.com/downloads/en/DeviceDoc/AVR-Instruction-Set-Manual-DS40002198A.pdf#_OPENTOPIC_TOC_PROCESSING_d138e25928) |  |

1. Imagine you are the author of a new Software Engineering textbook for high school students. You need to help students new to assembly language understand the operation of this code. Use a format such as an algorithm flowchart, an IPO chart or a text-based description to explain what is happening in the Assembly instruction code sample (Question 5 above). Assume that the microprocessor loops back to the start label while the result of the addition does not produce an arithmetic overflow.

|  |
| --- |
| Sample answers could include:  Initially, the value 1 is loaded into register 16. In a loop, the value in the register is output to a pin attached to the microcontroller (perhaps the speed of a motor or brightness of a display) then doubled. The loop continues iterating until the doubling produces an arithmetic overflow. So, the motor spins up to speed or the display fades up its brightness.  Sample IPO chart.Sample flowchart. |

1. Complete an equivalent algorithm in pseudocode in the space below.

|  |
| --- |
| Sample answer:  BEGIN  r16=1  REPEAT  OUTPUT r16 TO pin1  r16=r16+r16  UNTIL arithmetic overflow  END |

**Extension**: read more about opcodes and operands: <https://en.wikipedia.org/wiki/Opcode>

### The use of address and data registers

In the following activity, you will research processor registers and follow instructions on programming an Arduino using register manipulation.

**Activity 5**: Arduino Uno registers

Read the introductory paragraph on [Processor registers](https://en.wikipedia.org/wiki/Processor_register) to help answer the following questions.

1. Where are registers located and how are their contents modified?

|  |
| --- |
| Sample answer:  Registers are located in the processor and are modified via binary machine language commands.  The commands must be part of the processor’s instruction set. |

1. How are registers different from other types of computer memory, such as random access memory (RAM)?

|  |
| --- |
| Sample answer:  Registers are faster, more expensive and there are fewer of them. |

1. How do data registers and address registers differ?

|  |
| --- |
| Sample answer:  Data registers hold numeric data.  Address registers hold addresses used by machine code to locate the address of where to read or write data. |

1. Using the size of registers in your microcontroller’s microprocessor, show or describe how to calculate the maximum value each register can hold.

|  |
| --- |
| Sample answers could include:  For Arduino Uno: The Uno uses an 8-bit AVR processor. 8-bit registers can hold positive values between 0–255 (256 possible values). 2^8=256  For Pico: The Pico uses an ARM Cortex-M0+processor which is a 32-bit chip. 32-bit registers can hold positive values between 0–4,294,967,295 (4,294,967,296 possible values). 2^32=4,294,967,296 |

Read and follow the instructions in [Microcontroller Register Manipulation](https://www.instructables.com/Microcontroller-Register-Manipulation/) to understand the advantages of register programming and answer the following questions.

Use the [Arduino Uno registers simulation](https://wokwi.com/projects/387233104005573633) to copy, edit and run the code samples provided as you read the associated explanations.

1. The first snippet of code associates variables for each LED and each button with pin numbers that aren’t available on an Arduino Uno or in the simulation. See code example below:

int led\_1 = 26; // : 34 : 5 : 20 : 4  
int btn\_1 = 30; // : 30 : 0 : 35 : 8

Show the code needed to set these variables to the pin numbers used in the simulation

|  |
| --- |
| Sample answer:  int led\_1 = 4;  int led\_2 = 5;  int led\_3 = 6;  int led\_4 = 7;  int btn\_1 = 8;  int btn\_2 = 9;  int btn\_3 = 10;  int btn\_4 = 11; |

1. The first snippet of code sets the mode of the button pins to INPUT and the mode of the LED pins to OUTPUT.

What changes to data structures were used to allow a single FOR loop to set all pins to the correct mode in the second code snippet?

|  |
| --- |
| Sample answer:  Instead of single integer variables for led\_1, led\_2 … and btn\_1, btn\_2 … an array of integers was used for the LEDs and another array of integers for the buttons. |

1. Explain 2 advantages of programming a microcontroller using data and address registers directly.

|  |
| --- |
| Sample answer:  The compiled code can be smaller as it doesn’t need to use as many libraries as a normal program.  This can make it faster to execute as well as use less of the limited storage space on the microcontroller.  Using registers can also allow you to code extra functionality that isn’t normally exposed by the normal high-level code or available libraries such as changing the timing of PWM pins. |

## Identify and describe a range of sensors, actuators, and end effectors/manipulators within existing mechatronic systems

Mechatronic systems process inputs from their environment, such as from button presses or sound, and produce physical output such as gripping or moving things.

In this section, you will explore a range of electronic devices that collect input and produce output.

****Think about how humans and other animals’ sense, process and act on information received from our environment.

* How do we collect input?
* How do we process information?
* How do we produce output?

**Activity 6**: sensors and actuators

As a class, watch a video describing [sensors and actuators (2:35)](https://www.linkedin.com/learning/top-10-skills-for-robotics-engineers/familiarity-with-sensors-and-actuators).

Discuss examples of sensors and actuators found in common devices and complete the questions and tables below.

1. What are sensors and actuators responsible for in mechatronic systems?

|  |
| --- |
| Sample answer:  Sensors are responsible for providing input from the environment to a mechatronic system for processing before producing output.  Actuators are responsible for converting electrical signals into physical motion such as rotation or pushing/pulling. |

In small groups, watch one of the following student mechatronic project videos to help answer the question and table below.

* [Self-Playing Guitar (3:34)](https://www.youtube.com/watch?v=3a5GoUoeDbY)
* [“Micro Jackson Dancing Robot (3:12)](https://www.youtube.com/watch?v=r6Tb2ztNC40)
* [Coin Sorting Bank (4:50)](https://www.youtube.com/watch?v=K6YKykkaJEo)
* [Nerf Ball Gun Turret (3:02)](https://www.youtube.com/watch?v=S_RruxnvONQ)
* [Automated Greenhouse (2:27)](https://www.youtube.com/watch?v=MziF7lOGO-U).

1. Identify and describe the use of a range of sensors and actuators in your chosen system.

|  |
| --- |
| Sample answers could include:  Self-playing guitar – Sensors: switches and buttons; Actuators: servos to pluck each string, stepper motors to move ‘fingers’ to different frets, LEDs, and LCD display for status information.  Micro Jackson – Sensors: sound/audio sensor, buttons; Actuators: Servos and a DC motor to move body parts, LED eyes, LCD display, audio speaker.  Coin sorter – Sensors: pressure sensitive touchscreen, buttons; Actuators: DC motor to move goal indicator up/down, LEDs for status output, screen, piezo buzzer as an alarm, solenoid to lock door, photocell.  Nerf ball turret – Sensors: camera, switches; Actuators: display, servos to move turret and the balls, fly wheels to help launch the balls.  Greenhouse – Sensors: temperature sensor, light sensor, anemometer to measure wind speed; Actuators: piezo electric buzzer and LCD display for status information, servo to open or close roof, LEDs for night lights. |

1. In a small group, brainstorm different potential applications of sensors and actuators in different industries. [Think-Pair-Share](https://app.education.nsw.gov.au/digital-learning-selector/LearningActivity/Card/645): Record your group’s ideas below then compare and combine answers with other groups to complete the table.

|  |  |  |
| --- | --- | --- |
| Industry | Uses for sensors | Uses for actuators |
| Manufacturing |  |  |
| Healthcare |  |  |
| Agriculture |  |  |
| Construction |  |  |
| Transportation |  |  |

### Inputs – transducers and sensors

An electronic transducer is a small device that converts energy from one form into electricity.

The conversion uses no separate signal processing, and the resultant output signal is often too small for many microcontrollers to use directly.

Different transducers can convert sound, pressure, temperature, capacitance, torque and other environmental factors to electricity.

A sensor is a device that uses an appropriate transducer to measure an environmental quantity and convert it into an electrical signal appropriate for use in common microcontrollers.

Sensors are generally connected directly to a microcontroller or microcontroller board which can be programmed to read in the sensor value and process it with logic to determine the next appropriate actions.

**Activity 7**: sensors in mechatronic systems

1. As a class, watch [Sensors explained (8:52)](https://www.youtube.com/watch?v=gizihSJ63o4).

Brainstorm ways different sensors could be used in mechatronic systems and record your ideas below.

You could use sensors shown in the video or ones available in your school.

|  |  |
| --- | --- |
| Sensor | Suggested use in a mechatronic system |
|  |  |
|  |  |
|  |  |

### Motion sensors

As a class, watch [Motion sensors (1:45)](https://www.youtube.com/watch?v=SJDE36an3xk) and discuss ideas for their use in a mechatronic system.

Sensing motion in a mechatronic system can be done in a variety of ways, depending on the requirements of the system and the type of sensors available.

Some sensors output a signal to say whether motion has occurred.

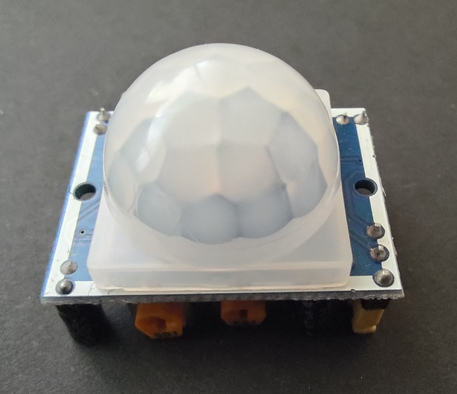
Some output a signal to say what motion has occurred. Some output a signal that requires the execution of extra logic in the microprocessor to determine what motion has occurred.

#### Passive infra-red (PIR) sensors

PIR sensors monitor infra-red radiation emitted from people and animals moving in their field of vision.

They are commonly used in automated lighting and alarm systems and are easily available in consumer packages for 240V mains power connections in hardware stores.

Figure 2 – PIR sensor



**Activity 8**: motion sensing with a passive infra-red sensor

Use this [Motion sensing with PIR simulation](https://www.tinkercad.com/things/19GUtBCEtNs?sharecode=ruxIGep0oDyvWAWDytihpXMMmDPx--38WOmaUCtB_KU) to investigate how a PIR sensor operates. You can login using your DoE education account to ‘tinker’ and modify the circuit. Select the ‘Code’ tab and add the following C++ code below before simulating.

int PIRpin=5;

void setup()

{

Serial.begin(9600); //Allows serial communication via the USB cable

}

void loop()

{

if(digitalRead(PIRpin)==true) Serial.println("Movement detected");

else Serial.println("No movement detected");

delay(100); // Wait for 100 millisecond(s)

}

After adding the code, simulate movement by clicking on the PIR.

The PIR sensor in the simulation identifies whether movement has been detected by showing 2 red lights.

Review the code and the serial monitor to see how the programming logic uses the data and creates some debug output to help answer the following questions.

1. The PIR indicates that movement has been detected for a period of time after movement in the detection window has stopped occurring.

Suggest a possible use for this functionality in a real-life scenario and whether the delay for detected movement would need to be shortened or lengthened.

|  |
| --- |
| **Sample answer:**  In a lighting system we often want a light to turn on and stay on for a short period of time after motion has been detected – for example, someone at the front door at night.  It is possible that motion would only be detected some of the time, so having the motion detector output a ‘motion detected’ signal for a time makes it easy to connect a light to the detector.  This way, no extra logic would be required to decide when to turn the light on and off. |

1. Investigate the binary selection control structure that controls the debug output.

The code calls an inbuilt function called digitalRead with the pin number that the PIR sensor is connected to as a parameter.

Based on your reading of the code, what data type does the function digitalRead return?

|  |
| --- |
| **Sample answer:**  The IF statements compares the output of digitalRead (PIRpin) to the value true. Therefore, the digitalRead function must return a Boolean value of true or false. |

1. There are 3 cables attached to the PIR sensor.

The simulation shows which pin each is connected to and the code references the pin used to collect the data from the sensor.

Identify the purpose for each jumper wire used in the table below.

|  |  |
| --- | --- |
| Wire colour used in simulation circuit | Wire connection purpose |
| Black |  |
| Red |  |
| Green |  |

#### Accelerometers

Accelerometers measure acceleration (changes in speed) over one, 2 or 3 directional axes.

Small accelerometers can be used in mobile phones to measure the phone’s movement in space and help determine how it is being used or to track physical activity.

They use piezoelectric transducers that convert physical movement into electrical signals.

When connected to a sensing circuit, useful outputs can be produced that can tell you how much acceleration the sensor has detected.

Simple accelerometers sense motion along a single line, dual-axis accelerometers sense motion on a 2D plane while triple-axis accelerometers can sense motion separately on all 3 axes (x, y & z).

**Activity 9**: accelerometer axes

As a class, watch the video [Accelerometer on micro:bit](https://www.youtube.com/watch?v=byngcwjO51U) (5:47).

Discuss ways that accelerometers with differing numbers of axes could be used to sense motion in mechatronic systems.

Collect and record the results below.

|  |  |
| --- | --- |
| Number of axes | Suggested uses |
|  |  |
|  |  |
|  |  |

**Extension**: read more about accelerometers: <https://en.wikipedia.org/wiki/Accelerometer>.

#### Gyroscopic sensors

Watch this video showing how [gyroscopic motion can be used to help code web pages (2:03)](https://www.youtube.com/watch?v=b_dTHSQVOZM).

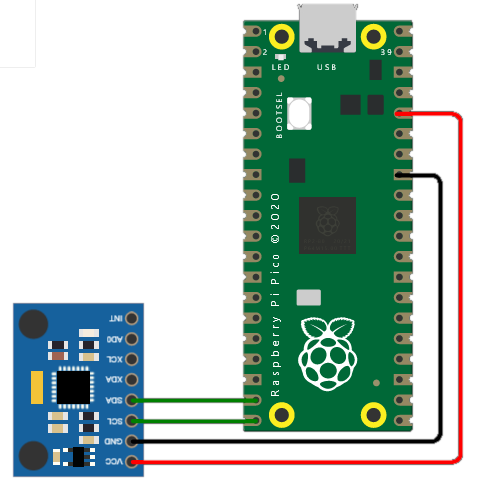
****Think about how a mobile phone knows which way it is being rotated when playing a game, or how you might feel during (or after) coming off a ride with a high degree of angular motion.

How does a mobile know that it is being moved correctly when calibrating the compass?

**Activity 10**: motion sensing with gyroscope

Use this [motion sensing with gyroscope simulation](https://wokwi.com/projects/383951631173430273) to verify that the code detects rotational motion around the x-axis.

Figure 3 – Pico and gyroscopic sensor



When simulating, click on the sensor to simulate motion.

1. Describe how the code used in the main program loop detects changes in rotational motion. The relevant Python code from the loop is copied below:

if(gx!=imu.gyro.x):

print(imu.gyro.x-gx, "degrees rotation change detected”)

gx=imu.gyro.x

|  |
| --- |
| **Sample answer:**  In the loop, a binary control structure compares the value coming directly from the sensor from the previously saved one.  If the values are different, it prints a message showing the difference and updates the saved value. |

1. Extend the code in the main program loop to detect motion around the y and z axes.

Copy your completed code from the page below.

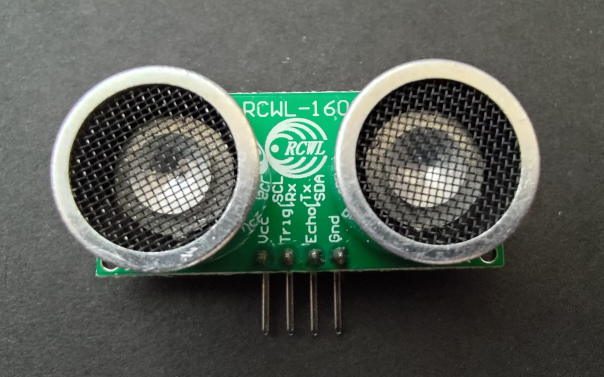
**Challenge:** Extend the code to detect motion using the accelerometer values as well.

|  |
| --- |
| **Sample answer:**  from imu import MPU6050  from time import sleep  from machine import Pin, I2C  i2c = I2C(1, sda=Pin(14), scl=Pin(15), freq=400000)  imu = MPU6050(i2c)  #get the initial x, y & z rotational values  gx=imu.gyro.x  gy=imu.gyro.y  gz=imu.gyro.z  while True:  if(gx!=imu.gyro.x):  print(imu.gyro.x-gx, "degrees rotation change detected around the x-axis")  gx=imu.gyro.x  if(gy!=imu.gyro.y):  print(imu.gyro.y-gy, "degrees rotation change detected around the y-ayis")  gy=imu.gyro.y  if(gz!=imu.gyro.z):  print(imu.gyro.z-gz, "degrees rotation change detected around the z-azis")  gz=imu.gyro.z  sleep(0.1) |

#### Ultrasonic distance sensors

Ultrasonic distance sensors only measure distance to an object in front of them but can be combined with some simple program logic to check for motion by checking for changes in distance.

Figure 4 – ultrasonic sensor



Measurements are taken by sending a short pulse of ultrasonic sound and then measuring the time taken to receive the echo of that sound. Program code is generally required to send the pulse and receive the echo as well as to convert the time to centimetres or another unit of distance measurement.

****Think about whether this is like the way people estimate the distance to a storm by measuring the time between seeing lightning and hearing the thunder.

Can you estimate the distance to a cliff face or across a valley by shouting and then timing the wait for the echo?

Watch this short [Ultrasonic Distance Sensor video (2:00)](https://www.youtube.com/shorts/O5l7IOmnD2Q).

**Activity 11**: ultrasonic distance sensor simulation

Use this [Motion sensing with ultrasonic distance sensor simulation](https://www.tinkercad.com/things/8WTJMFgO0IS-motion-sensing-with-ultrasonic-distance-monitor) to view and interact with a motion-detecting circuit using an ultrasonic distance sensor.

Copy in the code for the main loop shown on the following page.

void loop()

{

distance = readUltrasonicDistance(10, 10);

if (distance != lastDistance) {

Serial.print("Movement detected:");

Serial.print(abs((distance - lastDistance)));

if (distance<lastDistance) {

Serial.println("cm closer");

} else {

Serial.println("cm further");

}

lastDistance = distance;

}

delay(100); // Wait for 100 millisecond(s)

}

When simulating the circuit, select the ultrasonic distance sensor to simulate movement.

Select the ‘code’ button to view the code and then the ‘Serial monitor’ button to view the debug output.

1. Suggest a reason why the ultrasonic sensor cannot sense an object outside of the detection window shown in the simulation.

|  |
| --- |
| **Sample answer:**  If the object is too far away or at too great an angle, not enough sound would be reflected back from the object to the microphone to make a reliable detection. |

1. Inspect the serial monitor’s debug output window during simulation.

Describe how the code manages to output whether movement occurs and in which direction, even though it is not measuring motion directly.

Reference the control structures used in your response.

|  |
| --- |
| **Sample answer:**  At the start of each loop, the code reads the distance from the sensor and assigns it to the variable *distance*.  A binary selection control structure checks if the distance is different to the previous distance.  If it is, a calculation is made to determine the change in distance and then a nested binary selection is used to decide if the distance is closer or further than the previous one.  The *lastDistance* variable is updated. |

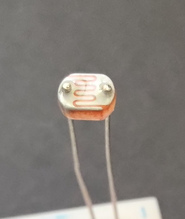
#### Light level sensors

There are a range of transducers and sensors that can detect light and produce an electronic signal.

Simple transducers change their electrical properties such as resistance, voltage or current depending on the amount of light they are exposed to.

Light sensors include some basic circuitry around these transducers to produce useful electrical signals that can more easily be used by a microcontroller.

Figure 5 – LDR



Some examples of transducers that sense light include:

* Light dependent resistors (LDRs)
* These reduce their resistance as they are exposed to more light.
* Watch these videos
* [What is an LDR? (3:30)](https://www.youtube.com/watch?v=2fvXW4OEWLE)
* [Using a piezo with an Arduino & photoresistor (0:33)](https://www.youtube.com/watch?v=Cok-FRz_-vc).
* Phototransistors
* These are based on transistors that have an input that is sensitive to light instead of an electrical voltage
* Watch [Phototransistor – Collin’s lab notes (0:59)](https://www.youtube.com/watch?v=5HKvRrVWgYs) describing a phototransistor circuit.
* Photodiodes
* These are more basic transducers than phototransistors.
* Photodiodes are best used in combination with a transistor to help amplify the output signal as part of a light sensing circuit.
* They are often employed in line-following sensors by detecting the amount of infra-red light received from surface reflection of an infra-red LED. This reduces the effects of ambient light in the room on a mechatronic system.
* Watch [Infrared LED & Photodioide (0:15)](https://www.youtube.com/shorts/3CvMyydiAlg) to see a photodiode detecting infra-red light.

**Activity 12**: light level sensing simulation

In each simulation linked below, use the 3 different light transducers provided to investigate how they are affected by light levels.

* Open the [light level sensing](https://www.tinkercad.com/things/4Yq9jDAaCkI?sharecode=w4PDMQTPzGxZ9cgso-bEShIZMJI0rK67yqlfIjWLlEQ) simulation.

Investigate how light levels can be used to change the speed of a motor for each of the 3 different light sensing transducers.

* Open the [light level plotting](https://www.tinkercad.com/things/gWbFZoMwhnE?sharecode=kxnHu4hLMedNnYfom93991g_HgikKLTFraCgyh1CLfI) simulation and add in the main loop code below.

Monitor the Serial output or Serial graph to examine the sensitivity of each of the different light sensing transducers.

Determine which transducers have a linear response and which have a non-linear response to changes in light.

A response is linear if it rises and falls proportional to the changes you make to the light falling on the transducer in a consistent way.

void loop()

{

Serial.println(analogRead(inputPin));

delay(100); // Wait for 100 millisecond(s)

}

1. Use the table below to record the motor speeds achievable with each light sensing device in the given circuit.

Record whether each transducer’s response is linear or non-linear.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Minimum motor actuator speed | Maximum motor actuator speed | Linear or non-linear response? |
| Light dependent resistor |  |  |  |
| Phototransistor |  |  |  |
| Photodiode |  |  |  |

1. Describe a potential use for a light level sensor in a mechatronic system.

|  |
| --- |
| **Sample answers could include:**  A light sensor could be used to detect a paper jam in a printer by having the paper pass between a light source and a matching sensor.  A light sensor could be used to detect the movement of a part in a machine if it had a series of holes that intermittently allowed or blocked the passage of light as the part moved.  A light sensor could be used to detect distance changes in an object by measuring the changes in light intensity reflected off an object from light shone at it. |

**Extension**: watch [Electret mic teardown (0:59)](https://www.youtube.com/shorts/P7YQRhHPasM) to learn how microphones sense sound.

****Think about whether it would be possible to use light level sensors or sound sensors to detect motion.

How would you go about it?

What issues might you have and how could you overcome them?

### Outputs – actuators and end effectors

Actuators and end effectors provide the outputs for movement of mechatronic systems.

Actuators are motors that produce rotational or linear movement and, in some cases, may be all that is required for parts of a system.

When more complex movement is desired, end effectors are used to convert the basic movement of an actuator, or a collection of actuators, to create more complex movement.

Examples of end effectors include a robotic gripper and movement in a mechatronic prosthesis.

**Activity 13**: actuators and end effectors

Find out what types of actuators or end effectors are available at your school.

Brainstorm different ways that these could be used in a mechatronic system.

**Teacher note**: if you have access to a 3D printer, this model for a [Snap-Together 9g Servo Gripper](https://www.thingiverse.com/thing:2775857) could be used as a valuable addition to this activity.

It uses a micro servo and doesn’t require any extra hardware, apart from the screws that come with the servo.

|  |  |
| --- | --- |
| Actuator or end effector | How this device could be used in a mechatronic system |
|  |  |
|  |  |
|  |  |
|  |  |

#### Rotary and linear actuators

Rotary actuators are motors designed to produce rotational motion while linear actuators use rotary actuators to produce linear motion by constraining the motion of parts connected to it.

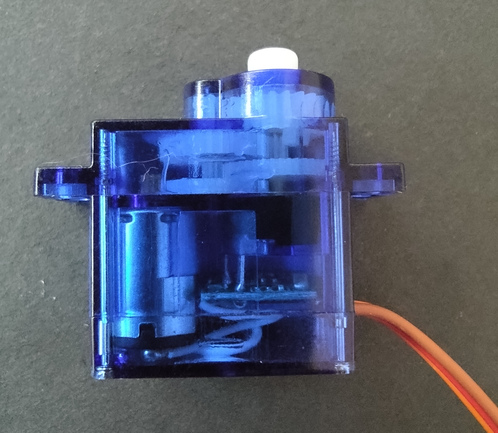
Many rotary and linear actuators need more current to operate than a microcontroller board can supply.

Connecting anything larger than a micro servo directly to a microcontroller board may permanently damage the board.

To control larger devices investigate the use of relays.

* DC motors – these are analog devices that increase their speed as the applied voltage increases.
* Servos – these are rotational actuators that are controlled by short pulses of voltage to set an approximate positional angle or speed.

Figure 6 – micro servo



* Positional servos – small voltage pulses control their position, but their range of angular motion is limited, so they are useful in mechatronic systems that require repeated angular motion. Watch the short video [Servo motors (0:58)](https://www.youtube.com/shorts/TfakrB1IzbM).
* Continuous rotation servos – small voltage pulses control their speed and can continuously rotate so can be used to continuously rotate light weight objects.
* Watch the short video [Continuous servos (0:59)](https://www.youtube.com/shorts/BuLn8khjGV8).
* Stepper motors – the internal electromagnetic motor design of stepper motors allows very precise rotational movements to specific preset angles so these are used when very accurate rotational motion is needed.

Read [Stepper motors and Arduino](https://howtomechatronics.com/tutorials/arduino/stepper-motors-and-arduino-the-ultimate-guide/).

**Activity 14**: rotary and linear actuator research

**Teacher note**: if you have access to a 3D printer, there is a printable and simple linear actuator model on [Thingiverse](https://www.thingiverse.com/thing:3170748) with options for fitting to either a standard or micro servo.

Watch [DIY Linear Servo Actuator](https://www.youtube.com/watch?v=2vAoOYF3m8U) (6:05) for a matching video on its assembly and use with an Arduino.

Use the Servo pulse width control simulation on [Tinkercad](https://www.tinkercad.com/things/4zclfwr5M96-pwm-servo-pulse-width-control?sharecode=FgsEyH5eo53VFrgOGOM-9vjxBaNbDvpu_PtPM2NuVA4) or [Wokwi](https://wokwi.com/projects/382497346497462273) to investigate the effects of changing the pulse width of a signal sent to a servo.

1. Find an example of each type of rotary and linear actuator, recording an image or screenshot of it.

Brainstorm with your class peers to suggest and record a possible practical use for each type in a mechatronic system.

|  |  |  |
| --- | --- | --- |
| Actuator | Image or screenshot | Possible use in a mechatronic system |
| DC motor |  |  |
| Micro servo (positional) |  |  |
| Micro servo (continuous) |  |  |
| Stepper motor |  |  |
| Linear actuator |  |  |

**Extension**: read more about [rotary actuators](https://en.wikipedia.org/wiki/Rotary_actuator) and [linear actuators](https://en.wikipedia.org/wiki/Linear_actuator).

### Hydraulic actuators

Watch the video of a [Hydraulic robot arm (1:08)](https://www.youtube.com/watch?v=3MR_O3xDAVU) in action and the video [How does a hydraulic cylinder work (4:00)](https://www.youtube.com/watch?v=52IMMQSB9Hs).

Hydraulic actuators move liquids through hoses to deliver the liquid to a destination or to use it to help power components of end effectors from a distance.

The incompressibility of liquid means that it can be pushed from a narrow hose into a wide hose to act as a force multiplier at the other end, or from a wide hose to a narrow one to act as a distance multiplier.

In each case, using a hydraulic actuator allows you to position the motor that provides the energy required to push the liquid at a distance from the end effector.

This is useful in multi-segment end effectors such as robotic arms which may not have the space or load capacity to hold heavy motors within its structure.

**Activity 15**: hydraulics research

Watch this video on [How hydraulic & pneumatic actuators work (4:07)](https://www.youtube.com/watch?v=95Y-WnbpQyk) and read about [hydraulics](https://www.explainthatstuff.com/hydraulics.html).

1. Use the space below to explain the operation of hydraulic actuators.

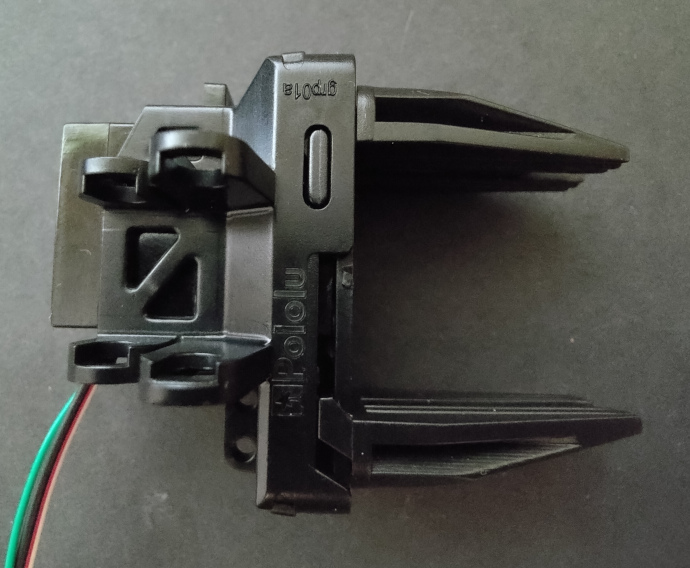
|  |
| --- |
| **Sample answer:**  Hydraulic actuators use the flow of liquids to create movement.  Liquid is pumped into a chamber connected to a piston rod.  As the chamber fills, the piston rod is pushed outwards.  Fluid can be pumped into the other side of the chamber, creating pressure in the reverse direction.  If the liquid in the first chamber is allowed to escape (collected in a reserve tank), the piston rod will be forced back inward.  By controlling the direction and flow of liquid, you can control the direction and speed or force of the piston rod. |

### Robotic grippers

Robotic grippers are end effectors designed to hold things, often to move them from one place to another.

Different types are suited to different purposes, but include suction, electromagnetic, hydraulic, hand and finger control.

Figure 7 – gripper



**Activity 16**: grippers and end effectors

**Teacher note**: if you have access to a 3D printer, this model for a [Snap-Together 9g Servo Gripper](https://www.thingiverse.com/thing:2775857) could be used as a valuable addition to this activity.

It uses a micro servo and doesn’t require any extra hardware, apart from the screws that come with the servo.

1. Read about robotic grippers on the links provided and complete the table to describe 4 different types of gripper.

* [Robot Grippers Explained](https://www.universal-robots.com/blog/robot-grippers-explained/)
* [Types Of Robot Grippers And Their Applications](https://dorna.ai/blog/types-of-grippers-for-robots/).

Watch [this video (1:31)](https://www.youtube.com/watch?v=yI8E50Orkng) showing the mechanisms used in a range of parallel jaw grippers.

|  |  |  |  |
| --- | --- | --- | --- |
| Gripper type | How it works | Practical application | Image or screenshot |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Imagine you were shown a mechatronic output device but that you didn’t know what its function was.

What are some features you would look for to determine whether the device was an actuator or an end effector?

|  |
| --- |
| **Sample answer:**  An actuator would have an output shaft that rotates, or a rod that moves in and out, but without connecting to something else that could usefully do something with that motion. An end effector would contain a motor, or have gearing or another connection, to connect to a motor or piston.  An end effector would likely have more visible moving parts than an actuator. |

**Extension**:

* Read more about [sensors](https://en.wikipedia.org/wiki/Sensor), [actuators](https://en.wikipedia.org/wiki/Actuator) and [end effectors](https://en.wikipedia.org/wiki/Robot_end_effector).
* Read [How to control LEDs on the Arduino](https://www.circuitbasics.com/how-to-control-leds-on-the-arduino/).

**Activity 17**: input/output revision

Classify each of the following as input or output, as well as transducer, sensor, actuator or end effector.

|  |  |  |
| --- | --- | --- |
| Device | Input or output? | Transducer, sensor, actuator or end effector? |
| DC motor [[example](https://core-electronics.com.au/hobby-motor-gear.html)] |  |  |
| A device that uses a DC motor and a specially-designed gearing system to clamp onto things tightly [[example](https://core-electronics.com.au/robot-gripper-extra-width.html)] |  |  |
| Accelerometer to measure movement on each of 3 different axes [[example](https://core-electronics.com.au/adxl335-triple-axis-accelerometer-gy-61.html)] |  |  |
| Photodiode – an unpowered device that converts incoming light to a small electrical signal [[example](https://core-electronics.com.au/mini-photocell.html)] |  |  |
| 180° positional servo – a device that turns a shaft with up to 180 degrees of rotation [[example](https://core-electronics.com.au/feetech-fs90-1-5kgcm-micro-servo-9g.html)] |  |  |

## Use different types of data and understand how it is obtained and processed in a mechatronic system, including diagnostic data and data used for optimisation

**Teacher note:** students should simulate circuits as well as construct them physically from this section onward if possible.

Consider setting the simulation components as homework tasks so that more class time can be spent on physical circuit construction.

The examples and activities provided are grouped by the type of data a device uses to facilitate device substitution so you can work with sensors and actuators that are available in your school.

The examples and activities may have different pin configurations to the microcontroller boards available at your school.

Students may benefit from a comparison with the pin configuration of the boards they will be using.

As a class, watch this video on connecting [I/O devices and peripherals (3:01)](https://www.linkedin.com/learning/iot-foundations-operating-systems-fundamentals/i-o-devices-and-peripherals?contextUrn=urn%3Ali%3AlearningCollection%3A7140549818396880896&u=74950778) to microcontrollers.

****What types of data were discussed in the video?

What protocols or methods for communicating with devices were mentioned in the video?

Can you think of any other types of data or communication methods that could be used in a mechatronic system?

As a class, discuss how analog and digital data differ.

Consider the benefits and limitations of each as well as the process of converting between them.

1. How are analog and digital data different?

Include in your response benefits and limitations of each type and how to convert between them.

|  |
| --- |
| Sample answer:  Analog data is data that can vary continuously between a range such as temperature, pressure, speed, time, voltage, current, height, weight and frequency.  It can have infinitely small variations that may not be perceptible.  Computers can sample analog data to the nearest available discrete binary value using an analog to digital converter (ADC).  Analog data is great for representing accurate values with incredible accuracy but requires advanced circuitry to deal with it as it is very susceptible to electrical noise and interference.  Digital data is data that can only have discrete values and can be represented exactly in the binary number system using only 1s and 0s.  Digital data is ideal for processing in a CPU as its transistors operate in an on/off mode so small fluctuations in voltage levels make no practical difference.  The process of sampling from analog to digital values, however, is only as accurate as the number of bits the ADC allows.  Sampling takes time and may introduce problematic delays into the collection of data in a system. |

In this section you will **prototype and build circuits** to test using data with your available sensors, actuators and microcontroller board.

The section is grouped into devices by the type of data used by a device.

When building and testing the circuits shown, you may need to substitute devices shown for other ones in the same category.

Review the basics of each of the following, where needed, before building your circuits:

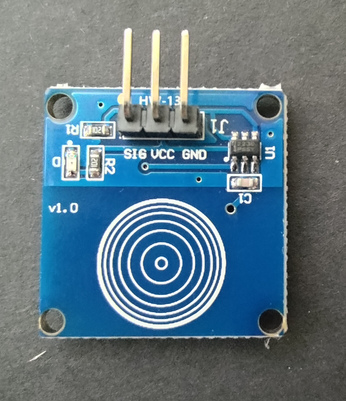
* Analog and digital data: <https://learn.sparkfun.com/tutorials/analog-vs-digital>
* Voltage, current and resistance <https://learn.sparkfun.com/tutorials/voltage-current-resistance-and-ohms-law>.

### Analog sensor data

Watch the video [Understanding analog to digital converters (1:50)](https://www.linkedin.com/learning/learning-arduino-interfacing-with-analog-devices/understanding-analog-to-digital-converters?contextUrn=urn%3Ali%3AlearningCollection%3A7140549818396880896&standalone=true&u=74950778).

Analog sensors produce a continuously varying output voltage signal that can be measured by an analog pin on your microcontroller board.

Figure 8 – analog sensor



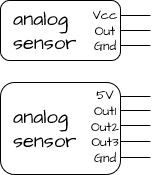
The measurement passes the sensor’s output voltage through an analog to digital converter (ADC).

This value can then be read and processed by program code with a special read command specific to the microcontroller used by your microcontroller board. ADCs in microcontrollers typically have 8–16 bits of resolution (256–65536 possible values), so storing ADC output data as an integer will work, although you may wish to consider unsigned byte or similar data types to reduce memory overhead.

Many sensors have 3 pins – one each for power, ground, and the analog output voltage.

The pins can be positioned in any order and can also have slightly different names.

Figure 9 – analog sensor pinouts



The diagrams above show a standard 3-pin sensor configuration as well as another analog sensor that produces 3 separate output signals (for example a 3-axis accelerometer).

Common pin names for each purpose are:

* Power – Vcc, Vss, Vin, 5V, 3.3V, 3-5V, +
* Ground – Gnd, -
* Analog output signal – Vo, Vout, Out, Sig, D.

**Teacher note**: this may be a good opportunity to brainstorm ideas for developing a class-wide colour-coding standard for wires.

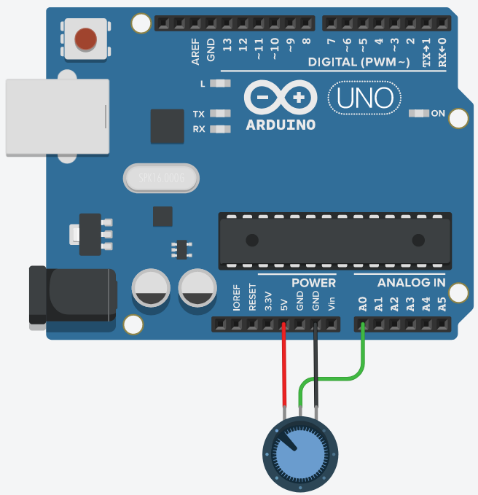
Examples could include red (+), black (-) and other colours for data signals.

Consider the complexities introduced when using a mix of voltage levels in circuits (for example 3.3V mixed with 5V) and devices that come pre-wired with different standards (such as servos).

**Activity 18**: collecting analog sensor data

Modify the analog sensor data simulation on [Tinkercad](https://www.tinkercad.com/things/lw55awHDnqt?sharecode=-UZS-oQW6k1jH5IEBgzz9jMsTfcXYg9oeqs1D0r76ho) or [Wokwi](https://wokwi.com/projects/382497327363044353) to determine the range of available values.

Figure 10 – Arduino Uno and potentiometer



Add the code below into the main loop to read and monitor the analog signal.

Monitor the range of outputs in the Serial monitor when using each voltage level and answer the following question.

**Arduino/C++ code**:

int analogValue=analogRead(A0); //Read value from pin A0

Serial.println(analogValue); //Show value in Serial monitor

delay(100); // Wait 100ms before starting next iteration

**Pico/Python code**:

analogValue=potentiometer.read\_u16() #Read value from pin

print(analogValue) #Show value in monitor

sleep(0.1) #Wait 100ms

Find the command required to read analog input data from an analog input pin with a chosen language on your microcontroller.

Record the command below.

|  |
| --- |
| **Sample answer could include:**  For Ardunio/C++: analogRead(A0);  For Pico/Python: potentiometer.read\_u16() |

1. Complete the table below for your chosen microcontroller or simulator.

|  |  |  |  |
| --- | --- | --- | --- |
| Minimum value | Maximum value | Number of bits of resolution in ADC | Calculate 2ADC bits for example 212 |
|  |  |  |  |

1. How does the number of possible values available from the ADC relate to the number of bits in each sample from the ADC?

|  |
| --- |
| **Sample answer:**  The number of possible values is equal to 2 to the power of the number of bits in each sample from the ADC.  For example, an 8-bit ADC has 28=256 possible values; a 10-bit ADC has 210=1024 possible values; a 16-bit ADC has 216=65536 possible values. |

**Extension**: after monitoring the output, change the wiring in the simulation to connect the potentiometer to the 5V pin instead of the 3.3V pin.

View the Serial monitor again and note how the range of detected values changes.

Explain the relationship between the voltage level provided to an analog sensor and its available range of analog output values.

|  |
| --- |
| **Sample answer:**  If the voltage level provided to the ADC matches the full logic level, the value returned will be the full value (1023 for a 10-bit ADC, 65535 for a 16-bit ADC).  When the voltage level is lower, the value returned is a fraction of the full value.  If the microcontroller uses 5V logic, but the potentiometer’s power pin is connected to a 3.3V source, the highest value will be limited to 3.3/5 multiplied by the normal maximum. |

Construct a physical circuit that reads analog data from a sensor and outputs the current value from the ADC. Copy your code and an image of the circuit below.

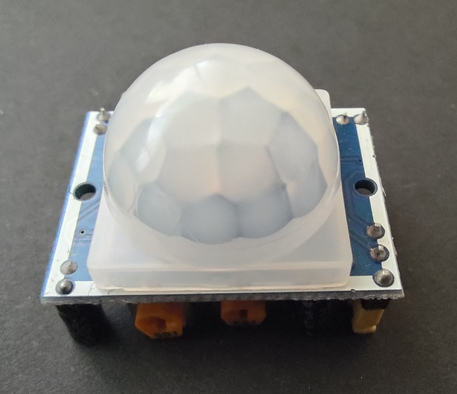
**Challenge:** Modify the code to only produce output when the analog value has changed from the previous reading.

|  |  |
| --- | --- |
| Code to read sensor value and produce output | Image of circuit |
|  |  |

### On/off digital sensor data

On/off digital sensors come with similar pinouts to analog sensors, but the output is either a high or a low voltage.

Figure 11 – digital sensor



They are useful when the presence (or absence) of something needs to be indicated rather than a range of values describing something about the detection.

On/off digital data is most efficiently stored using a Boolean data type and can also be used directly within control structure conditions.

****Think about how lift doors know whether it is safe to open or close, how a security light decides when to turn on or how a mobile phone knows whether to keep the display on during use.

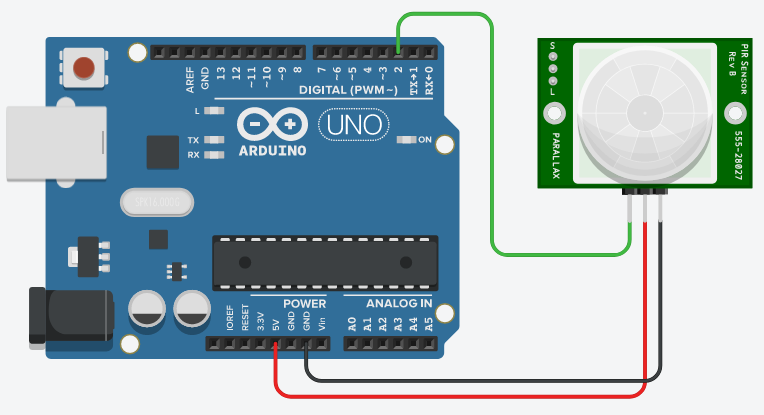
What on/off information could these devices be using to make these decisions?

**Activity 19**: collecting digital sensor data

Modify the digital sensor data simulation on [Tinkercad](https://www.tinkercad.com/things/a3araTY082x-digital-sensor-data?sharecode=P4qKVqwGbk-sB3vowueUarlirk9eZE1VEU-eR9PaDXM) or [Wokwi](https://wokwi.com/projects/382497311139473409) to connect a PIR motion sensor to the microcontroller board.

The sensor needs power and should be connected to a digital pin used by the code.

Figure 12 – Arduino Uno and PIR sensor

****

**Arduino/C++ code:**

void setup()

{

pinMode(2, INPUT); //Use digital pin 2 for input

Serial.begin(9600); //To allow viewing in the Serial monitor

}

void loop()

{

boolean digitalValue=digitalRead(2); //Read value from digital pin 2

Serial.println(digitalValue); //Show value in Serial monitor

delay(100); //Wait before ending iteration

}

**Pico/Python code:**

from machine import Pin

from time import sleep

PIR=Pin(28,Pin.IN) #Configure pin 28 for digital input

while True:

digitalValue=PIR.value() #Get the value at the pin

print(digitalValue)

sleep(0.1)

Modify the code and circuit to use a different microcontroller pin.

Use the Serial monitor to verify that the circuit still works.

Then copy your code and an image of your circuit into the table below.

**Extension**: connect the PIR motion sensor to an analog input and adjust the code to check whether the value being read from the analog to digital converter is above or below a threshold value.

|  |  |
| --- | --- |
| Code to read sensor value and produce output | Image of circuit |
|  |  |

Find the command(s) required to read digital input data for a digital input pin with a chosen language on your microcontroller.

Record the command below.

|  |
| --- |
| **Sample answers could include:**  **For Arduino/C++**  pinMode(3, INPUT);  int value=digitalRead(3);  **For Pico/Python**  PIR=Pin(0,Pin.IN)  Value=PIR.value() |

Construct a physical circuit that reads digital data from a sensor and creates an appropriate output. The output could be a buzzer or LED that turns on and off, movement of a servo or Serial data to be monitored on a connected computer.

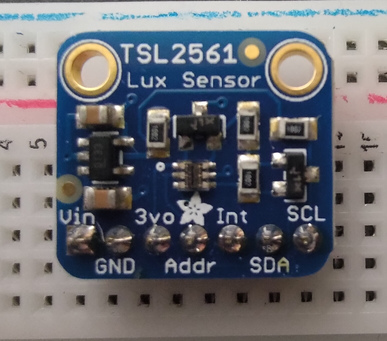
Copy your code and an image of the circuit below.

|  |  |
| --- | --- |
| Code to read sensor value and produce output | Image of circuit |
|  |  |

### Digital I2C sensor data

When you want a microcontroller to receive multiple bytes of data in a single read operation, Boolean data would be inappropriate.

Figure 13 – I2C light sensor



You could design a circuit with multiple wires to send data bits in parallel, but it would take up many pins on a microcontroller board and significant space in a physical circuit.

There are multiple communication protocols that support the sending of bytes of digital data including Serial and SPI, but I2C is becoming the most common due to its ease of use.

**Activity 20**: I2C protocol research

**Teacher note**: the simulation in this activity is only available on Wokwi.

Watch [Introduction to I2C (3:50)](https://www.youtube.com/watch?v=qeJN_80CiMU).

1. Read about the I2C protocol on the links provided and complete the table below.

* [Inter-Integrated Circuit (I2C) Protocol](https://docs.arduino.cc/learn/communication/wire)
* [I2C](https://learn.sparkfun.com/tutorials/i2c).

|  |  |
| --- | --- |
| Prompts | Response |
| Three advantages of using I2C |  |
| What 4 wires are always required to connect an I2C device? |  |
| Line of code that retrieves I2C data from a device |  |
| How does an I2C device know if a command from the microcontroller is intended for it? |  |

Use the [IMU data simulator](https://wokwi.com/projects/383981627952028673) to monitor data coming from a device connected using the I2C protocol.

Add the following line to use a library file included with the simulation:

from imu import MPU6050

Add the following line to instantiate a MPU6050 object called imu that uses the i2c variable as a parameter in its constructor:

imu = MPU6050(i2c)

Record a screenshot showing a successful plot of multiple changing values from the I2C sensor.

|  |
| --- |
| Sample answer:  A screen shot of a sample output graph. |

### Combined sensor data

Some sensors output multiple types of data.

A good example is a joystick with directional controls as well as buttons. The directional controls produce an analog output for each axis (for example x, y). The state of each button can be provided as a digital on/off signal for each push button on the controller.

Figure 14 – joystick sensor



Some sensors output the same set of data using multiple protocols such as analog as well as I2C.

Using sensors that output multiple types of data can be useful when you are not sure which types of input pins your microcontroller board will still have available.

### ‘Analog’ outputs – pulse width modulation

Analog output devices such as LEDs and DC motors are designed to take an analog input signal to drive them.

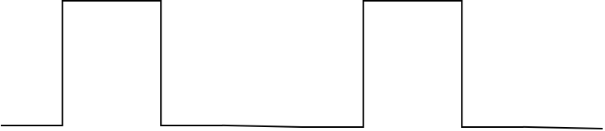
A higher input signal produces a brighter or faster response.

Most microcontrollers, however, do not have the ability to produce actual analog output signals.

They can only output digital signals of high and low but can do this very quickly.

By adjusting the width of a repetitive signal’s pulse, microcontrollers can mimic an analog signal. This is called pulse-width modulation (PWM) and its availability is generally indicated on microcontroller board pins using the symbol ‘**~**’ (tilde). PWM can be used to drive analog devices as shown in Figure 15.

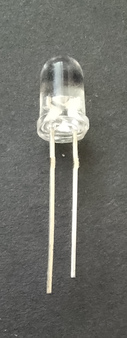
Figure 15 – a PWM signal with a pulse about 1/3 the width of the repeated waveform. This can mimic an analog signal 1/3 the voltage of the pulse voltage



### LED

A light emitting diode (LED) is a 2-pin device that produces light when current flows through it in the correct direction.

Figure 16 – LED



Current should be limited by placing a resistor in series at either end of the LED.

This avoids damaging the LED as well as the microcontroller.

**Activity 21**: analog (PWM) LED control

**Teacher note**: the Wokwi simulator doesn’t simulate LED brightness correctly for PWM signals, so this simulation is provided only on Tinkercad.

Modify the [PWM LED control simulation](https://www.tinkercad.com/things/lN4mABxxL6g?sharecode=TvqH-En4GxxRZDA36rRu1G52NzKjXg_R4EYyVSxXK7A) to do the following:

* Configure pin 3 as an output pin in the setup() subprogram.

Pin 3 is the one providing a PWM signal to the LED.

* pinMode(3, OUTPUT);
* Increment the brightness variable in each iteration of the loop.
* If it gets past its maximum value of 255, reset it to 0.
* Increase the brightness of the LED by changing the value of the resistor.

**Challenge:** Use the RGB LED to produce an output that fades between each of its 3 colours in turn.

1. Copy your code and an image of the circuit below.

|  |  |
| --- | --- |
| Code to control RGB LED through PWM | Image of circuit |
|  |  |

Find the command syntax required to produce ‘analog’ PWM output on a pin with a chosen language on your microcontroller.

1. Record the command(s) below.

|  |
| --- |
| **Sample answers could include:**  **For Arduino/C++**  pinMode(3, OUTPUT);  analogWrite(3,127);  **For Pico/Python**  LED=PWM(Pin(0))  LED.freq(200)  LED.duty\_u16(32767) |

Construct a physical circuit that outputs a pulse-width modulated analog signal to an analog device to verify that it works correctly with your microcontroller board.

1. Copy your code and an image of the circuit below.

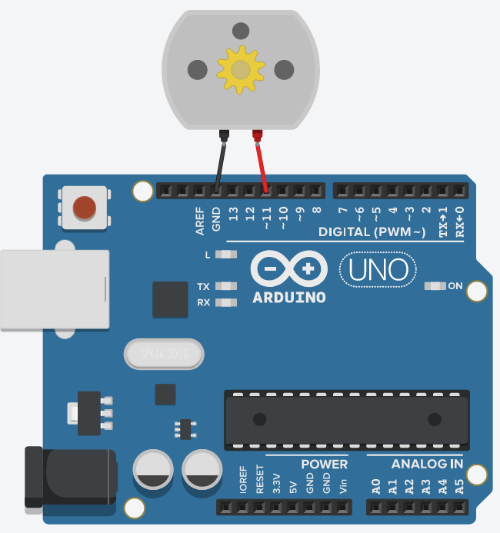
**Challenge:** Create a physical circuit to fade through the colours of an RGB LED.

|  |  |
| --- | --- |
| Code to control analog device through PWM | Image of circuit |
|  |  |

### Motor

DC motors increase their speed as the analog voltage across their terminals increases.

Figure 17 – Arduino Uno and DC motor (note: model only)



It is unlikely that your physical microcontroller board can safely control a DC motor without damaging its internal circuitry as too much power is required to move the motor shaft, so we will just prototype code for it on TinkerCad in this activity.

**Activity 22**: motor control simulation

**Teacher note**: the Wokwi simulator does not provide a DC motor, so this simulation is provided only on Tinkercad.

Modify the [PWM motor control simulation](https://www.tinkercad.com/things/0PnkYS1yzkn?sharecode=kMRgNjICPwgxJ7w3l23uylng8jpdIxGLt_zm8p4adiA) to more safely control the motor.

The code currently resets the speed to 0 after reaching its maximum of 255.

This quick change has the potential to burn out the motor and damage the microcontroller board. After the motor increases its speed slowly, make it decrease slowly as well.

Sample pseudocode has been provided in the table below.

Note that you should **never connect a motor directly to a microcontroller board.**

There is not enough power to drive the motor from the controller’s pins.

Connecting a motor this way **can only be done in the simulator**.

|  |  |
| --- | --- |
| Possible pseudocode |  |
| REM Global variables  speed=0  speedChange=5  BEGIN loop()  OUTPUT speed to a PWM pin  IF speed<0 or speed>=255 THEN  speedChange = -speedChange  END IF  speed=speed+speedChange  END |  |

### Servo

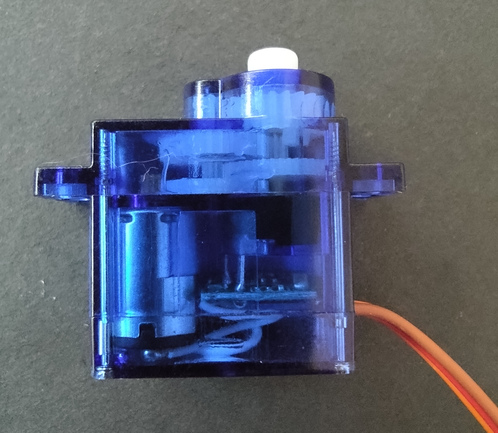
**Teacher note**: servos are described here for coding without any external libraries. Adapt to your servo control library if appropriate.

Servo control libraries are discussed in Servos (p.168) It is recommended to use micro-servos for this activity. Servos can draw considerable currents and may not work or could damage the microcontroller board if they are too large or heavily loaded.

Most microcontroller boards will tolerate a very small number of lightly driven micro-servos, especially when connected directly to its pins. If using more than 2–3 unloaded micro-servos, or any larger servos, you will need to power them directly rather than from the microcontroller’s power pins.

Servos are small motors that can be more accurately controlled than DC motors, but they require a very specific PWM waveform.

Figure 18 – mircroservo



The waveform required is usually a pulse of between 1 & 2 milliseconds in a repeated signal with a frequency of 50Hz or less (a period of 20 milliseconds or more).

Continuous servos use the pulse width to set the desired speed, while positional servos use the pulse width to set the angle.

**Activity 23**: servo control

**Teacher note**: the Wokwi simulator doesn’t provide continuous rotation servos, so this simulation is provided only on Tinkercad.

A similar [simulation using a 180o servo on Wokwi](https://wokwi.com/projects/382497269926238209) is available for comparison.

**Step 1:** investigate the [Manual PWM rotational servo control simulation](https://www.tinkercad.com/things/86eSrMxNPDg?sharecode=7N6T6DVpwLiu9M3rlaX1xNK9kgoCo0h1hFVQcPRRGYM) to determine the range of pulse width values required to move the wheel in differing directions.

**Step 2:** add the code shown into the main control loop to create a pulse.

**Step 3:** simulate the circuit and view the Serial monitor and see the pulse widths being sent to the servo.

**Arduino/C++ code**:

digitalWrite(12, HIGH);

delayMicroseconds(pulseWidth);

digitalWrite(12, LOW);

delayMicroseconds(20000-pulseWidth);

**Pico/Python complete code**:

# PWM control of a servo motor without any external libraries

from machine import Pin

from time import sleep\_us

servo=Pin(0,Pin.OUT)

pulseWidth=1000

while True:

  #create a small high pulse followed by a longer low signal with a total period of 20ms

  servo.on()

  sleep\_us(pulseWidth)

  servo.off()

  sleep\_us(20000-pulseWidth)

  print(pulseWidth)

  pulseWidth=pulseWidth+1;  #change the increment to change the speed of the servo

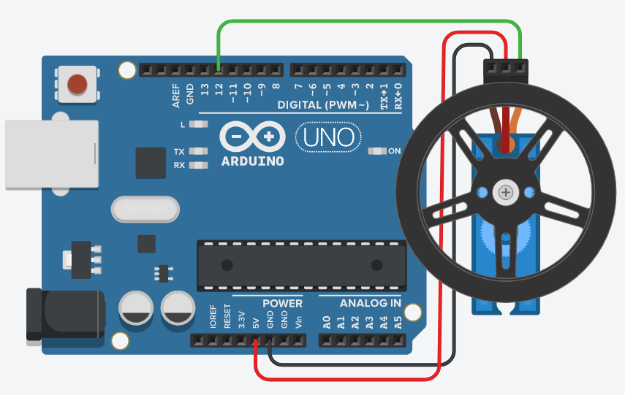
  if pulseWidth>2000:

    pulseWidth=1000

Complete the table below.

|  |  |
| --- | --- |
| Servo movement | Pulse width value or range |
| Clockwise |  |
| Stationary |  |
| Anticlockwise |  |

Figure 19 – Arduino Uno and continuous-rotation servo



Construct a physical circuit to move a servo motor in a controlled way such as increasing speed or sweeping through a range of angles.

Most servos use the following wire colours:

* Brown: ground (connect to a ground pin on your microcontroller board)
* Red: power (connect to a power pin on your microcontroller board; 5V if available)
* Orange: PWM signal (we are controlling this manually, so any digital pin will work)

Copy your code and an image of the circuit below.

|  |  |
| --- | --- |
| Code to control servo | Image of circuit |
|  |  |

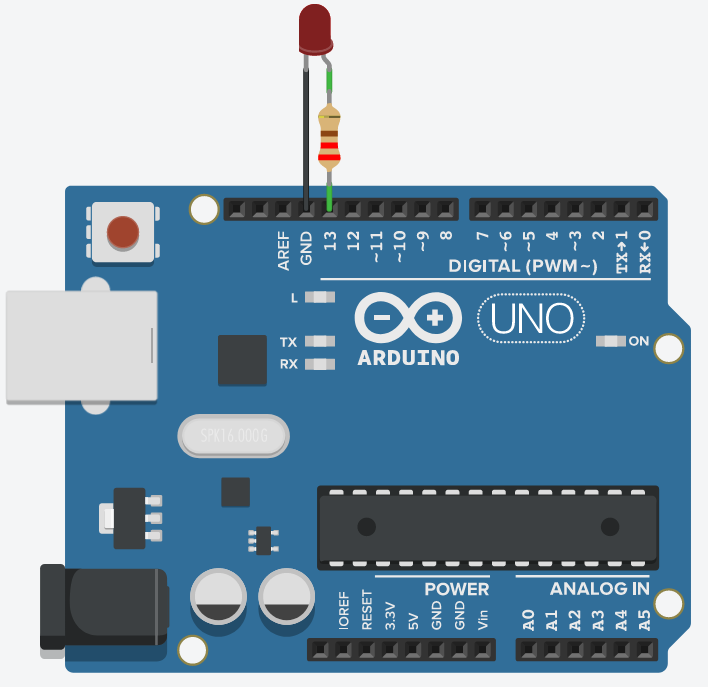
**Extension**: read more about [How to Control Servo Motors with Arduino](https://howtomechatronics.com/how-it-works/how-servo-motors-work-how-to-control-servos-using-arduino/)

### Digital outputs

Digital outputs produce high or low signal levels to represent a binary bit value.

This can be useful for binary control such as turning something on or off.

Figure 20 – Arduino Uno and LED with resistor



**Activity 24**: digital LED control

Use the Digital LED control simulation on [Tinkercad](https://www.tinkercad.com/things/bUvNVWQVqJB-digital-led-control?sharecode=0H2YU4cilmzjiKH_rk5TMv3OPQUXR6uaJ8HDqQkgk-s) or [Wokwi](https://wokwi.com/projects/382497549868810241) for this activity.

When complete, you should be able to control an LED to turn it on and off repeatedly.

**Step 1:** find and record below the command(s) required to produce a digital output on a pin on your microcontroller. Record the command(s) below.

|  |
| --- |
| **Sample answers could include:**  **For Arduino/C++**  pinMode(13, OUTPUT);  digitalWrite(13,HIGH);  **For Pico/Python**  LED=Pin(21,Pin.OUT)  LED.on() |

**Step 2:** combine your code from above with the pseudocode below in your simulation to toggle the LED on and off.

IF LEDon==true THEN

OUTPUT HIGH on the LED pin  
ELSE

OUTPUT LOW on the LED pin

END IF

DELAY 500 ms

LEDon=!LEDon //Reverse the status variable for the next iteration

**Extension**: modify the simulation to use an RGB LED so that it turns each of the 7 possible colour combinations on and off.

Connect the cathode to ground and each of the R, G and B legs through resistors to 3 different digital pins.

Record your code and an image of your circuit below.

|  |  |
| --- | --- |
| Code for digital control of LED | Image of circuit |
|  |  |

**Step 3:** construct a physical circuit that outputs a digital signal to a light emitting diode (LED) to verify that your code works with your microcontroller board.

The longer leg should be connected through a resistor to the output pin and the shorter leg should be connected to ground.

Watch this video to learn how to [calculate LED resistor value (0:56)](https://blog.adafruit.com/2021/09/28/new-video-calculate-led-resistor-value-collins-lab-notes-adafruit-collinslabnotes/).

If you don’t know the current required for your LED, assume 10mA (0.01A).

**Extension**: produce a range of colours by using an RGB LED instead.

Copy your code and an image of the circuit below.

|  |  |
| --- | --- |
| Code for digital control of LED | Image of circuit |
|  |  |

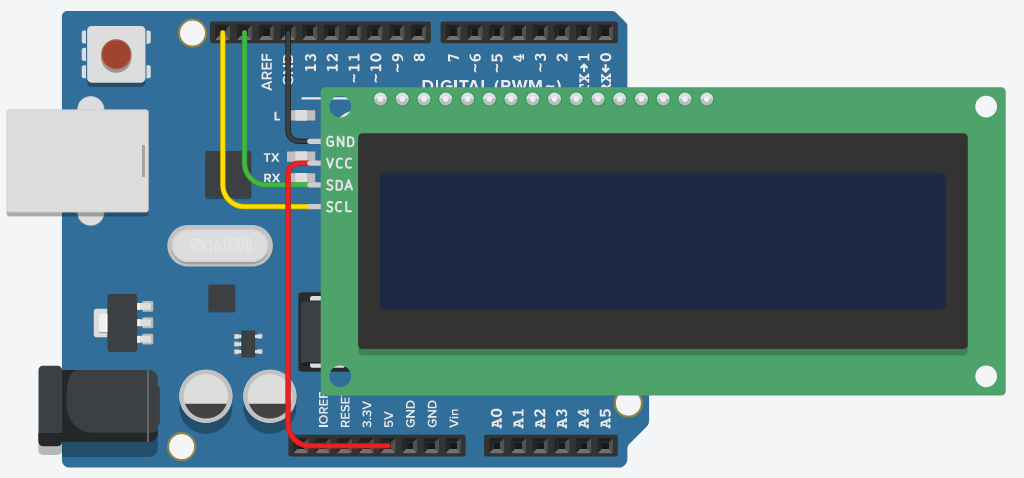
### I2C outputs

**Activity 25**: LCD simulation using I2C

Simulate controlling LCD text using I2C on [Tinkercad](https://www.tinkercad.com/things/8agRp6pPU7B-lcd-text-using-i2c?sharecode=XNiWChCGgtRp2QxrVRa894hb85FPVF-9Ni88gOHlC2Q) or [Wokwi](https://wokwi.com/projects/382497225412085761).

**Step 1:** include appropriate library file(s) to enable I2C communication with the LCD display.

Figure 21 – Arduino Uno and I2C LED display



**Arduino/C++ code**:

#include <Adafruit\_LiquidCrystal.h>

**Pico/Python code**:

from lcd\_api import LcdApi

from i2c\_lcd import I2cLcd

**Step 2:** send a hello message on startup.

**Ardiuno/C++ code**:

void setup(){

lcd.begin(16, 2);

lcd.print("hello i2c");

}

**Pico/Python code**:

lcd.putstr("Hello I2C")

**Step 3:** create a main loop with this code inside:

**Arduino/C++ code**:

lcd.setCursor(0, 1);

lcd.print(seconds);

lcd.setBacklight(1);

delay(500);

lcd.setBacklight(0);

delay(500);

seconds = seconds + 1;

**Pico/Python code**:

sleep(1)

lcd.clear()

lcd.putstr(str(seconds))

seconds=seconds+1

Insert a screenshot below showing your simulated working circuit.

|  |
| --- |
| Image of working circuit |
|  |

## Experiment with software to control interactions and dependencies within mechatronic systems

In this section, you will control mechatronic device outputs with an adjustable analog input.

The sample simulations provided use potentiometers, but you may choose to substitute these for other devices such as joysticks, LDRs (photoresistors), ultrasonic distance sensors or whatever is available at your school.

### Motion constraints

**Teacher note**: if you have access to a 3D printer, constructing or revisiting either of the following models may be a good addition to the following activity:

* [simple linear actuator](https://www.thingiverse.com/thing:3170748)
* [Snap-Together 9g Servo Gripper](https://www.thingiverse.com/thing:2775857).

Motion constraints in mechatronic systems include:

* purposeful design constraints to constrain the directions of movement
* Limits to available movement due to the nature or interaction of parts of a system.

**Activity 26**: constrained motion systems

Figure 22 – constrained motion inside gripper

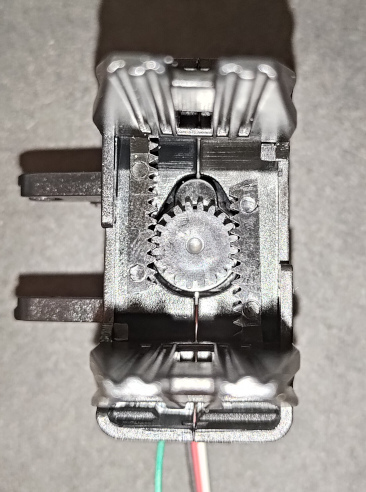
****

Figure 22 shows some internal components of a small robotic gripper.

The 2 fingers at the top and bottom of the image are connected to a horn on a servo motor via connected teeth.

As a class, watch [Constrained motion (3:27)](https://youtu.be/OQbrwBbWfqo) and discuss motion constraints in this system.

1. Describe the motion constraints in a simple 2-finger robotic gripper

|  |
| --- |
| **Sample answer:**  The fingers will only move as far as the servo and gears driving them. Additionally, the fingers can only move in and out as far as the body housing them mechanically allows. The motion of the fingers is constrained to move linearly by the gripper body and gear connections. |

Watch this [linear actuator animation (1:02)](https://www.youtube.com/watch?v=stOcZqdEjmE) and [video on electric linear actuators (3:53)](https://www.youtube.com/watch?v=e9V2GY1WaRU) showing how constraining motion from a rotary actuator can be used to create linear motion. Answer the questions below.

1. Explain how rotational motion can be used to create a linear actuator.

|  |
| --- |
| **Sample answer:**  Restricting the motion of a nut or bolt connected to a threaded rod on the output of a motor or servo can create linear motion.  If the nut/bolt is not allowed to turn, rotation of the threaded rod will force the nut/bolt to move in or out in a linear fashion. |

1. How can the speed or force in a linear actuator be adjusted?

|  |
| --- |
| **Sample answer:**  Gearing that connects the motor to the rotation threaded rod as well as the thread pitch on the rod. |

Watch these short videos and discuss the motion constraints of the devices shown in each:

* [Bambu lab 3D printer (0:33)](https://www.youtube.com/shorts/3Pz3xrujm5w)
* [Mechatronics vehicle project (0:38)](https://www.youtube.com/shorts/BKqxYLgQcDA).

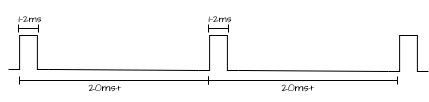
1. Describe the motion constraints for different parts of a mechatronic system of your choice.

|  |
| --- |
| Sample answers could include:  3D printer: The print head is constrained to move 2 axes, each controlled with a separate motor connected to a toothed belt.  The print head can also only move within the body of the printer itself.  Vehicle: The wheels can move forwards and backwards – using differing wheel motion on each side of the vehicle provides steering.  The arm consists of multiple sections each of which can extend and retract linearly.  The arm also has one section that can rotate about 30 degrees in a single plane. |

**Activity 27**: investigate servo motion constraints

Use the Servo pulse width control simulation on [Tinkercad](https://www.tinkercad.com/things/4zclfwr5M96-pwm-servo-pulse-width-control?sharecode=FgsEyH5eo53VFrgOGOM-9vjxBaNbDvpu_PtPM2NuVA4) or [Wokwi](https://wokwi.com/projects/382497346497462273) to measure the range of motion achievable in a simulated servo.

Figure 23 – PWM signal for servos



Measure or approximate the available angle range of the positional servo against the pulse widths provided.

You will need to view the Serial monitor to check the pulse widths.

Experiment by changing calculations in the source code to see if providing a greater range of pulse widths changes the achievable angle range.

Record your results in the table below.

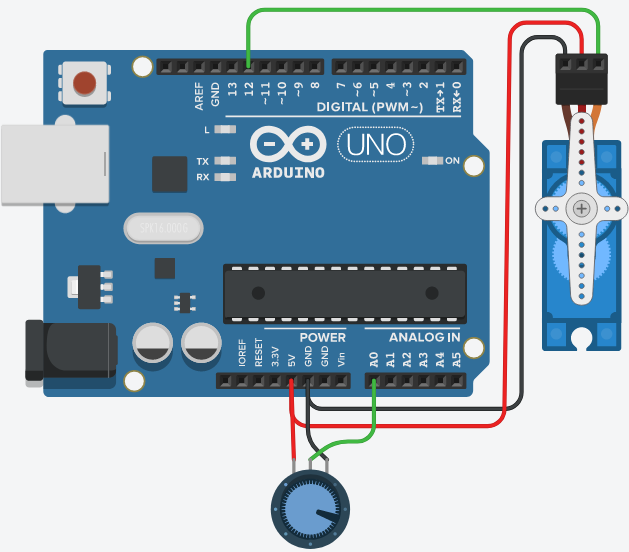
|  |  |  |
| --- | --- | --- |
| Type of code | Range of pulse widths (μs) | Motion range (degrees) |
| With original code |  |  |
| With modified code |  |  |

Construct a physical circuit to control and measure the motion constraints of a servo.

You will need to connect:

* a potentiometer or other analog input device to ground, power, and an analog input
* you could substitute this for a photoresistor, joystick, distance, or sound sensor and so on)
* a servo to a ground, power, and a digital output.

Figure 24 – Arduino Uno, potentiometer and micro servo



You will need to use code to:

* read from an analog sensor
* write to a digital output.

You may use the following pseudocode.

The code below assumes your microcontroller uses 10-bit ADC (values from 0-1023) and maps these to a range close to 500–2500 (the maximum range for most small servos). You may need to adjust these for your microcontroller, input device and servos.

BEGIN

CONFIGURE an analog pin as INPUT

CONFIGURE a digital pin as OUTPUT

WHILE true

REM Collect and scale a 10-bit ADC value

INPUT analogValue

pulseWidth=500+analogValue\*2

REM Produce a pulse of the desired pulse width

OUTPUT HIGH to servo

Pause execution for pulseWidth microseconds

OUTPUT LOW to servo

Pause execution for 20000-pulseWidth microseconds

END WHILE

END

Record the minimum and maximum pulse widths that 3 different servos respond to as well as the range of motion of each.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Minimum pulse width (μs) | Maximum pulse width (μs) | Angle range (degrees) |
| Servo1 |  |  |  |
| Servo2 |  |  |  |
| Servo3 |  |  |  |

### Degrees of freedom

The term ‘Degrees of freedom’ (DOF) refers to the range of motion of a mechatronic system.

The 2 common ways of describing this range of motion are:

* the final motion (or desired motion) of a single part or the end effector
* the total controllable motion across all parts of the system.

Watch this [2DOF motion simulation platform (1:03)](https://www.youtube.com/watch?v=ARU62KzuZM0) to see an example of a system with 2 degrees of freedom.

The platform simulates motion felt by the driver of a racing car from its acceleration and cornering.

The platform rotates in 2 different directions using 2 micro servos.

**Activity 28**: degrees of freedom research

When considering the available motion of a single part or end effector in a system, we consider both translational and rotational movement in each of 3 dimensions.

Read [What does Six Degrees of Freedom Mean?](https://www.techopedia.com/definition/12702/six-degrees-of-freedom-6dof) and watch [Six degrees of freedom (6DoF) explained (2:12)](https://youtu.be/LHzkwt513q0).

When considering the combination of controllable motion across the parts of a mechatronic system, we think about the total number of controllable aspects of its motion.

This can be more than 6 degrees of freedom if there are many interconnected parts. Watch [What is 6 Degrees of Freedom? (2:47)](https://www.youtube.com/watch?v=DdvBrKl3SHg).

1. Complete the table below describing the 6 degrees of freedom of a single object in space. Some parts have been completed for you.

|  |  |
| --- | --- |
| Term | Description |
| Pitch | Rotation around the Y-axis (between the X-axis and Z-axis) |
|  | Movement along the Y-axis |
|  |  |
|  |  |
|  |  |
|  |  |

Watch this video showing a [7-dof robotic arm (0:59)](https://www.youtube.com/watch?v=BZnzIHdAoGo) and discuss the advantages of having extra control over the positioning of the end effector.

1. Describe 2 advantages in motion control when a mechatronic system has more freedom in motion than strictly required for positioning of the end effector.

|  |
| --- |
| Sample answer:  Providing extra axes of motion can help create smoother movement of the end effector.  The end effector can stay in a fixed position while rest of mechanism moves around it.  This could possibly allow faster movement or obstacle avoidance on the mechanism’s planned path. |

Watch [8 DoF humanoid robot arm (0:22)](https://www.youtube.com/watch?v=iEeTZ0rD2yU) and [17 dof biped robot (3:27)](https://www.youtube.com/watch?v=n_1aVNnz9rA) to see examples of robots with more than 6 degrees of freedom.

1. Explain how the term ‘degrees of freedom’ is used to describe motion in a mechatronic system with interconnected moving parts.

|  |
| --- |
| Sample answer:  The term ‘degrees of freedom’ is commonly used to describe motion in 2 different ways: the number of degrees of freedom in space an end effector has (translation and rotation in each of 3 different axes) or the number of connected moving parts in a system. |

### Combination of subsystems

**Teacher note:** this section starts a guided project to help students create a small mechatronic game.

The project is split across 2 activities.

Suggested parts include:

* microcontroller board with connection to a power source
* jumper wires
* potentiometers x2
* piezoelectric buzzer x1
* light sensor (for example LDR) x1
* separate 5–6V power source if using larger servos, longer arms or planning to load the servos in any way.

Extra support files available in the [Software Engineering channel of the TAS Statewide Staffroom](https://teams.microsoft.com/l/channel/19%3Ac4aa94ee6a3340e3b35ca2f2c40375df%40thread.tacv2/13.%20Software%20Engineering%2011-12?groupId=cd5a04e1-7742-47dd-b141-9519486d9e00&tenantId=05a0e69a-418a-47c1-9c25-9387261bf991) for this activity and for the next activity include the following:

* videos showing a sample project in development and a completed project
* link to a simulated project on [wokwi.com](https://wokwi.com/)
* Python code for running the sample project on a Raspberry Pi Pico
* Adobe Illustrator file for laser cutting pieces using 3 mm material.

A subsystem is a part of a mechatronic system.

It can be a part of the hardware, a subprogram, or part of the software, or a combination of these such as a subprogram controlling a piece of hardware.

Subsystems can include:

* microcontroller board
* signal conditioning algorithms such as noise filtering
* control algorithms for smooth actuator movements
* sensors to gather inputs
* actuators to produce outputs
* a combination of sensors and actuators designed to operate together
* other connected devices such as hats, shields, and motor drivers.

**Activity 29**: combine sensors and servos

In this activity, you will be combining some initial components for a 2-player mechatronic game. The game will be completed over 2 activities.

When complete, one player will be controlling a moving target that has 2 degrees of freedom while the other player tries to shine light on it for points.

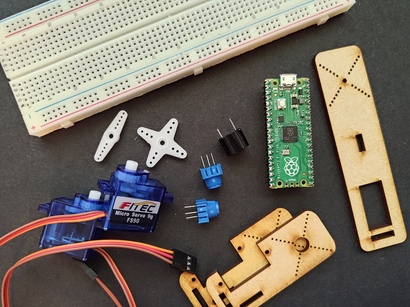
This first activity focuses only on controlling servos with potentiometers. You will be using a library to control the pulse width modulation required by the servos.

This will reduce the need to include specific timing code for different parts of the system.

Complete these steps in order to start making your game.

* **Step 1:** gather equipment.
* You will need a breadboard, microcontroller board, 2 servos, 2 potentiometers, a buzzer, and some jumper wires. You will also need something to mount the servos to each other.

Figure 25 – project components

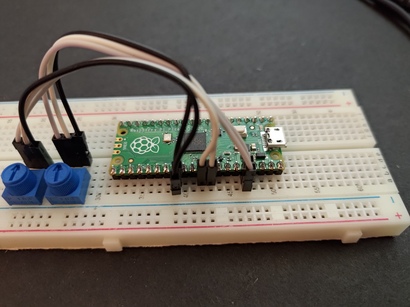


* **Step 2:** connect and test analog sensors.

When this step is complete, you should be able to see sample values from each potentiometer changing as you manipulate or change their position.

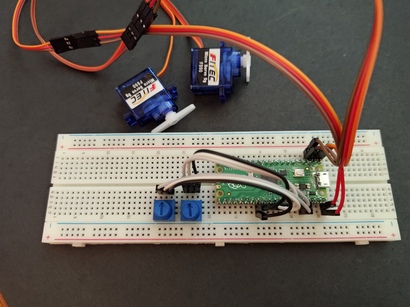
* Connect the potentiometers’ outside pins to the ground and power pins on your microcontroller board.
* Connect each analog output to an ADC pin on your microcontroller board.
* Sample [Tinkercad](https://www.tinkercad.com/things/duXYtpMv8qy-game-test-analog-sensors?sharecode=dYmC20cLwcMVFIoDqR75VNM44pZk5IZy5W8gL7e3MQg) and [Wokwi](https://wokwi.com/projects/382596416920309761) wiring diagrams are available but note that the wiring may be different from your devices.

Figure 26 – sample wiring with Pico



* **Step 3:** test your sensors, edit the code in the samples above or do the following:
* Set the pin mode for each ADC sensor pin to analog input.
* In a loop, read and print the value of each sensor pin. Add a small delay.
* Monitor the debug output to verify that the readings from both potentiometers span the full range of values from the ADC.
* **Step 4:** connect and test servos. When this step is complete, you should be able to control servo motion using code on your microcontroller.

Figure 27 – sample wiring including micro servos



* If using larger, faster or loaded servos, or using a robotic arm kit, this step will require you to use a separate motor controller board or a separate power supply for the servos.

This will safely handle power delivery to servos without damaging the microcontroller or its power source such as your laptop or battery.

* Connect each servo’s ground and power pins to ground and power pins on your microcontroller board or motor driver board.

Connect each servo’s PWM control pin to a PWM-capable digital pin on your microcontroller board.

See the sample [Tinkercad](https://www.tinkercad.com/things/bUYuyGgEijH-game-test-2-servos?sharecode=NsVVaIWtRDkhcSVZ_TzJueSxshMjzznhChvKA_M775o) or [Wokwi](https://wokwi.com/projects/382593928813933569) wiring diagrams if needed but note that the wiring may be different from your devices.

* To test your servos, edit the code in the samples above or do the following:
* Set the pin mode for each servo to output.
* Set the frequency of the servo signals to 50Hz (a servo library may do this for you).
* Loop through a range of values to create a pulse with of 1–2ms. In each iteration of the loop, send the pulse value to the servo. Add a small delay.
* Verify that each servo moves through a range of angles.
* **Step 5:** combine sensors and servos with control code.

When this step is complete, you should be able to control each servo by changing the position of a potentiometer.

* Create code to allow the analog sensors to control the servos.

One sensor or joystick axis will control one servo.

The other sensor or joystick axis will control the other servo.

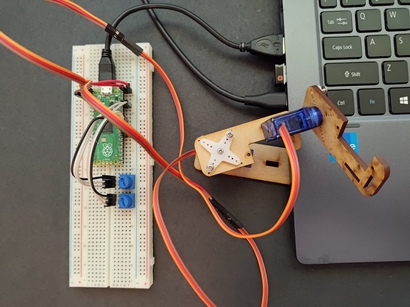
In a loop, perform these actions:

* Read each sensor value.
* Scale each sensor value to a range suitable to create a pulse width of 1–2ms.
* Send the pulse value to each servo.
* Add a small delay.
* Sample code for the loop is available on [Tinkercad](https://www.tinkercad.com/things/4NGqzYWvs3I-game-test-sensors-and-servos-combined?sharecode=ZpNWgnpts4i5eTDKpXANsIZsqWHrZm_YKcyW0JHkdXc) and [Wokwi](https://wokwi.com/projects/382599566948786177).
* Verify that you can control the servos with the potentiometers.
* **Step 6:** attach the servos in a physical system to allow them to move in 2 dimensions.

Some methods to achieve this include:

* using a laser cut template to fit servos into
* attaching the servos directly together. The horn from the first servo can be attached to a block of wood. The horn from the other servo can be attached to the first servo

Figure 28 – sample connection to arms



* creating a structure out of cardboard
* using a pan and tilt kit. These can be purchased or printed on a 3D printer
* using a robotic arm kit. If using a robotic arm kit, you must use a separate motor driver board or power supply to handle the larger currents required by the servos.

**Extension 1**: create a function to scale the reading from the analog pin to a value suitable for your servo. Edit the code to call the function instead of performing the calculations directly in the main loop.

**Extension 2**: edit your code to avoid the need for a servo or PWM library. Create pulses of the correct duration and spacing for each servo.

1. Record your code and an image of your working circuit below.

|  |  |
| --- | --- |
| Source code | Image of circuit |
|  |  |

1. How does using a servo library or PWM library change the nature of the code when combining servos or other subsystems in a mechatronic project?

|  |
| --- |
| Sample answer:  Using a library to control PWM pulses to servo pins removes the need to maintain special pulse timing code in the main program loop.  It simplifies the code especially when combining other subsystems that may have their own delays or timing requirements. |

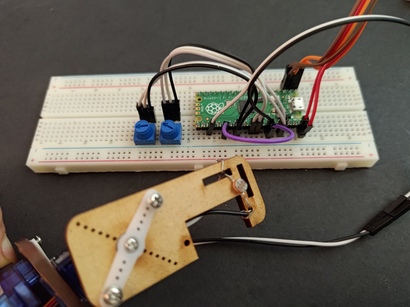
### Combination of sensors, actuators and end effectors to create viable subsystems

**Activity 30**: complete sensor and servo game

In this activity, you will add a light sensor and a buzzer to your system to create a working game. When complete, one player will control the servos while another tries to shine a light directly at the moving light sensor.

Once enough light has been sensed, a winning sound will play.

Figure 29 – adding LDR



Follow these steps to complete your game

1. Connect and test a light dependent resistor (LDR) or similar analog light transducer.

When this step is complete, you should be able to see sample values changing based on the amount of light being received by an LDR.

1. Attach an LDR to the game’s arm.

You may need to stick or cut and twist the ends of a jumper lead to each leg of the LDR.

1. Connect the jumper wires to your microcontroller board using a voltage divider configuration.

Watch this [Voltage divider (0:59)](https://www.youtube.com/watch?v=lQnDAs-igQM) video to understand how voltage dividers work. The LDR forms one resistor in the voltage divider.

You may be able to use an internal resistor on your microcontroller with a pin configuration command to avoid using an extra resistor in your voltage divider.

If using an Arduino board, you can use INPUT\_PULLUP as a parameter to the pinMode command.

If using a Pico, you need to manually connect the ADC pin to another digital IO pin configured as Pin.PULL\_UP. See examples below.  
Arduino/C++: pinMode(A5,INPUT\_PULLUP);  
Pico/Python: Pin(20,Pin.IN,Pin.PULL\_UP)  
 LDR=ADC(Pin(28)) #Connect pin 28 and 20 together

Use the sample circuit and code on [Tinkercad](https://www.tinkercad.com/things/ePgpSwoZmBl-game-test-ldr?sharecode=JT0Wt29ntvtGpmO7FpT0u-tk-nJ8vIViwiFHwpH9Ww8) or [Wokwi](https://wokwi.com/projects/383244023092493313) to help set up your circuit and code.

Note that this Wokwi sample uses a sensor board rather than a plain LDR.

Your LDR connection may need to be slightly different.

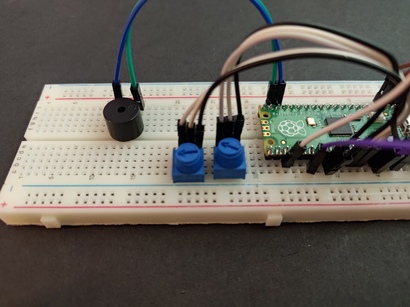
The circuit pictured uses an LDR connected to pins 28 and ground.

It also has a wire connecting pins 20 and 28 directly.

1. Verify that your circuit can sample values from the LDR’s voltage divider.
2. Connect and test a piezoelectric buzzer.

When this step is complete, you should be able to play different frequencies through a piezoelectric buzzer.

Figure 30 – adding buzzer



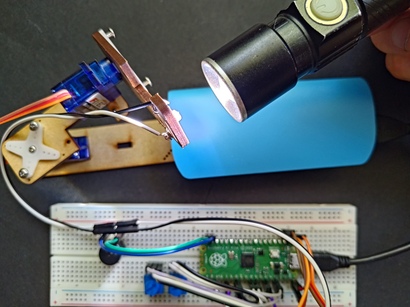
1. Attach a buzzer to your microcontroller board. Attach its positive pin to a PWM-capable IO pin and its ground pin to a microcontroller ground pin.
2. Configure the IO pin as a PWM output pin, for example:  
   Arduino/C++: pinMode(9,OUTPUT);  
   Pico/Python: buzzer=PWM(Pin(14))
3. Use the sample circuit and code on [Tinkercad](https://www.tinkercad.com/things/1beKthYgagf-game-test-buzzer?sharecode=uuJayPBWwG4pJnziEL8Py7zLQtNFK3_fcZVXlnTh-DQ) or [Wokwi](https://wokwi.com/projects/383247063303557121) as a guide or edit to make it work with your system.
4. Run the code and verify that you can hear beeps of increasing frequencies.
5. Code and test the game.

When this step is complete, you should be able to play the 2-player game.

One player should be able to control the servos while the other attempts to shine a light directly on the LDR.

A winning sound should be heard when enough time has elapsed with strong light falling on the LDR.

Figure 31 – completed game



1. Use the sample [Tinkercad](https://www.tinkercad.com/things/aLDn5yK6n4O-game-complete?sharecode=zlaO-dWE3MdjyI9IQoszTdOtxgQxjUZilqE1d37eaXc) or [Wokwi](https://wokwi.com/projects/382604400571376641) circuit and code as a guide.

Edit the pin numbers to match your circuit or commands to match your language.

Edit the value in the binary selection that checks for strong light being sensed on the LDR to suit your sensor and room conditions.

1. Check the game’s functionality

Each potentiometer should control a single servo.

Shining a bright light on the LDR should make the buzzer sound and its frequency increase.

After enough time has elapsed with a bright light on the LDR, a winning sound should play and the game should start again.

1. Play the game with a partner.

Experiment with adjusting values or logic in the code to make winning challenging but achievable.

**Extension 1**: edit the winningSound() subprogram to create a more interesting sound.

It could be a loop that sweeps through a range of closely spaced frequencies in rapid succession.

**Extension 2**: use the level of ambient light at startup to determine the cutoff value to use when detecting whether torch light is landing directly on the LDR.

Consider sweeping the servos through a range of motion to check ambient light coming in from all directions.

**Extension 3**: automate computer control of the servos when nearby motion is detected through a PIR sensor.

Ensure each servo only moves in small increments to avoid damaging the servo, the microcontroller or device supplying power to the system.

Create a flowchart representing the logic in your main loop.

Record your code and the flowchart in the table below.

|  |  |
| --- | --- |
| Source code | Flowchart of main loop |
|  |  |

## Determine power, battery and material requirements for components of a mechatronic system

**Activity 31**: develop component specifications for sensor and servo game

Imagine that you’d like to manufacture and sell a mobile version of your sensor and servo game. To do so, you will need to generate a complete parts list and wiring diagram.

You will also need to ensure the device can be powered without tethering to a computer with a USB cable.

Find the technical details of each component required to manufacture your game in online electronics stores, recording the details in the table below.

Ensure that the servos are not powered from the pins of your microcontroller. The pins of your microcontroller will not safely power the servos which may be put under load in the hands of your customers.

To help determine the required capacity of a battery power source in milliampere-hours (mAh), multiply the total number of milliamperes required by your circuit (mA) by the number of hours you’d like the system to run for before recharge or battery replacement.

This can be expressed using the formula mAh=mA x h.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Component | URL | Price | Quantity | Supply voltage | Current required |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

1. Specify your required battery capacity in mAh below. State what other materials will be required for production of your sensor and servo game.

|  |
| --- |
| Sample answer:  5V battery capacity required=2000mA\*2h=4000mAh (4Ah)  A circuit board and solder to connect all electrical components  Arms and mounting mechanisms to hold the servos and light sensor in place |

## Develop wiring diagrams for a mechatronic system, considering data and power supply requirements

Watch the video on wiring diagram [schematics (6:10)](https://www.youtube.com/watch?v=9cps7Q_IrX0).

**Activity 32**: wiring diagrams

Using [NESA’s course specifications for Software Engineering](https://library.curriculum.nsw.edu.au/341419dc-8ec2-0289-7225-6db7f2d751ef/94e1eb0a-0df7-4dbe-9b72-5d5e0d17143a/software-engineering-11-12-higher-school-certificate-course-specifications.PDF), find the table showing symbols used for different devices in a wiring diagram (p 24).

Using this information and your own research, complete the table below to show the purpose and correct symbol for each type of component.

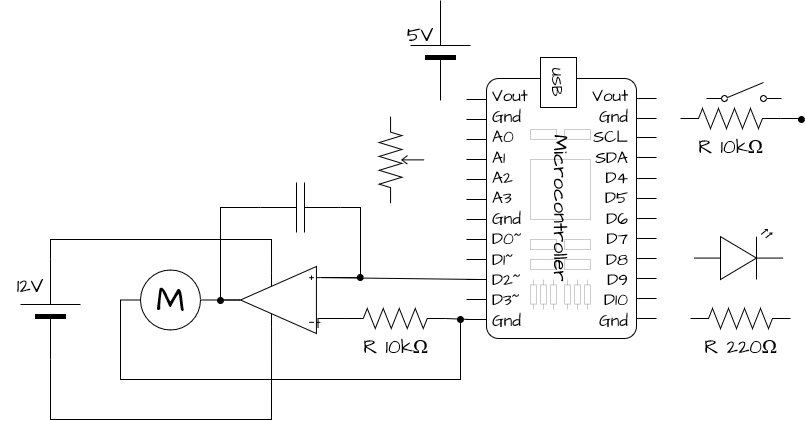
|  |  |  |
| --- | --- | --- |
| Component | Purpose | Symbol(s) |
| Capacitor |  |  |
| Diode |  |  |
| Resistor |  |  |
| Potentiometer or variable resistor |  |  |
| 2-way switch |  |  |
| On/off switch |  |  |
| Speaker |  |  |
| Motor |  |  |
| LED |  |  |
| Lightbulb |  |  |
| Integrated circuit |  |  |
| Voltage source |  |  |
| Operational amplifier |  |  |

An incomplete wiring diagram for a planned motor control circuit is shown below.

The section on the bottom left is a plan to connect a motor to a PWM pin through an amplifier to produce a signal from 0 to 12V with more current than the microcontroller can supply.

Other components are shown for you to power the microcontroller, connect a potentiometer and switch to control the motor, and use an LED to show the system status to the user.

Figure 32 – incomplete wiring diagram



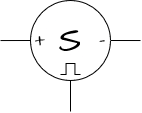
On the diagram connect components as below:

* Variable speed control of the motor with a potentiometer
* Connect one end of the potentiometer to power and the other to ground
* The tapped end in the centre of the potentiometer should connect to an analog input for reading its value
* On/off control of the system through the use of a switch
* Connect one end of the switch to power and the other to a digital input
* Add a connection to ground through a resistor from the end connected to the digital input. This resistor to ground connection prevents the value from ‘floating’ to an unpredictable value when the switch is open (off)
* User feedback through a light emitting diode (LED)
* Connect one end of the LED to a digital output and the other end to ground through a resistor. The resistor limits the current that can flow through the LED to avoid damaging it
* Power the microcontroller through its USB socket
* Use the USB socket to connect to a voltage source, but show the negative side of the voltage source connecting to ground.

|  |
| --- |
| Sample answer:  Sample completed wiring diagram. |

Using an appropriate diagramming tool, create a wiring diagram that represents the circuit you created for the sensor and servo game. Use a symbol similar to the one shown on the right for each servo.

Figure 33 – sample servo circuit symbol



|  |
| --- |
| Sample answer:  Sample complete wiring diagram. |

## Determine specialist requirements that influence the design and functions of mechatronic systems designed for people with disability

**Activity 33**: specialist requirements in mechatronic design

Watch [An affordable, refreshable braille tablet that relies on microfluidics (3:04)](https://www.youtube.com/watch?v=0fIg4rI4cDw).

The video shows a design for a refreshable braille system and mentions a variety of specialist requirements for readers with a visual disability.

1. Identify the specialist requirements of a refreshable braille reader for users with a visual disability.

|  |
| --- |
| Sample answer:  The device’s size, weight, affordability, portability and ability to display multiple lines of text as well as spatially-distributed information. |

****In a small group, brainstorm, and record ideas for how you could change the physical interface in your sensor and servo game for people with different types of disability.

1. Identify specialist design requirements for a game similar to your sensor and servo game for people with a specific type of disability.

In your response, state the type of disability and the specialist requirements.

Also suggest how the design might be modified and what to include [for example, for someone without manual dexterity to control knobs, the design could include a joystick, tablet or 3DOF feedback to assist control].

# Designing control algorithms

A control algorithm is code that controls the operation of a mechatronic system.

It uses values from the available sensors to compute outputs that it then sends to connected actuators to control the system’s overall state.

As a class, watch this video showing motion with differing [control of a line following robot (3:15)](https://youtu.be/t3R9cPq1aYk). Discuss the impact and potential design of each of the controlling algorithms.

A control algorithm typically has the following structure:

* Set-up or calibration. This may be needed to measure the range of sensor values in the system’s current environment. For example, a line following robot may need to measure the amount of reflected light from a line and from the background so that the algorithm can distinguish them.
* A main control loop that is repeated while the system is in operation. A typical iteration would perform the following:
* Read values from sensors to find the current state of the system. More sensors and more accurate readings will give a better picture as to the actual state of the system. Using more sensors adds to the cost and size of a system while reading from more sensors takes more time.
* Compute control values designed to help make the system approach a desired set point. A set point is a desired system state such as a room’s temperature or a car’s speed or direction. The microprocessor speed can be important if the calculations are complex.
* Output values to actuators to adjust or maintain the state of the system. A physical system takes time to respond and generally does not behave exactly as you might have modelled it. For example, when changing direction on a servo, it takes some time for the gears to mesh with each other again. When driving a wheel, the softness of the rubber slows the response given by the motor.

**Activity 34**: control algorithms

1. Describe how the motion of the robot differs across the 2 control algorithms used.

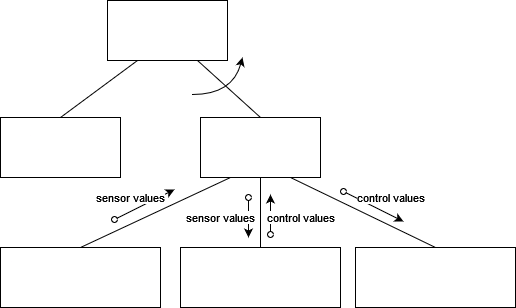
|  |
| --- |
| Sample answer:  The robot with the PID control algorithm moves slightly faster and more smoothly than the one using bang-bang control.  The robot with bang-bang control moves in a small straight line until it decides that it has gone ‘too far’ after which it makes an abrupt turn back in the right direction. |

1. Why does the operator move the robot at the start of the video?

|  |
| --- |
| Sample answer:  The operator is calibrating the robot.  Its calibration subprogram is reading the light reflected from the black and white surfaces so that the control algorithm more accurately knows what each sensor is above. |

1. Using the information about control algorithms, complete the structure chart shown to describe the basic structure of a control algorithm.

Figure 34 – incomplete structure chart



|  |
| --- |
| Sample answer:  Sample complete structure chart. |

## Develop, modify and apply algorithms to control a mechatronic system

**Activity 35**: modify sensor and servo game code

Recall the sensor and servo game you made previously.

You will now modify its code to control the servos so that the light sensor automatically avoids the light source.

This will change it into a single player game.

One technique you can use is to move each servo a small random amount when the detected light level is high enough for the player to score points.

You can try other techniques if desired. Copy your modified working code below.

**Extension**: make the automatic movements more controlled. Rather than using random values, maintain variables to record which direction each servo is being moved. Change one of these variables at random times to help avoid the light source.

|  |
| --- |
| Sample answers may include:  For Arduino/C++ ([Tinkercad simulation](https://www.tinkercad.com/things/hDRwBtaZDpT-game-automatic-light-avoidance?sharecode=FOR4RHhJPOjtU5doexn0EI_AtNutjGLlWiwFscTaUaw)):  //Completed game - with automatic light avoidance  #include <Servo.h>  Servo servo1, servo2;  int score=0; //Start game with no points  int servo1Value, servo2Value; //Control values to send to the servos  void setup()  {  //attach each servo to a PWM-capable pin  //the servo library sets the 50Hz pulse frequency  servo1.attach(5, 1000, 2000);  servo2.attach(6, 1000, 2000);    Serial.begin(9600);    reset();  }  void loop(){    int light=analogRead(A5);  if(light>50){ //Strong light detected - change this value to suit your room and LDR  servo1Value=servo1Value+rand()%20-10;  if(servo1Value<=0) servo1Value=0;  if(servo1Value>=180) servo1Value=180;  servo2Value=servo2Value+rand()%20-10;  if(servo2Value<=0) servo2Value=0;  if(servo2Value>=180) servo2Value=180;  servo1.write(servo1Value);  servo2.write(servo2Value);  score=score+10;  tone(9,score);  Serial.println(score);    if(score>1000){ //If the game has been won, play a sound and get ready for the next round  winningSound();  reset();  }    }else{ //Strong light not detected .. do nothing (turn the buzzer off)  noTone(9);  }  delay(100);  }  void reset(){  //Reset score variable and provide user feedback before a new game  score=0;  servo1Value=90, servo2Value=90; //Start servos in the middle  beep(2);  Serial.println("Ready for new game");  }  void beep(int numBeeps){  //Turn the buzzer on and off a given number of times  Serial.println("Beep");  for(int i=0; i<numBeeps; i=i+1){  tone(9,1000);  delay(300);  noTone(9);  delay(200);  }  }  void winningSound(){  //Play a nice winning sound  Serial.println("Winner!");  beep(3);  //add a better winning sound here - eg loop through a range of frequencies in quick succession  }  For Pico/Python ([Wokwi simulation](https://wokwi.com/projects/386516587027747841)):  #Completed game – with automatic light avoidance  from machine import Pin,ADC,PWM  from time import sleep  #Configure the servo pins for PWM output  servo1=PWM(Pin(0))  servo2=PWM(Pin(1))  servo1.freq(50)  servo2.freq(50)  #Configure the potentiometer pins for ADC input  potentiometer1=ADC(Pin(26))  potentiometer2=ADC(Pin(27))  Pin(20,Pin.IN,Pin.PULL\_UP) #50k resistor to 3.3V for voltage divider with LDR transducer  LDR=ADC(Pin(28)) #Configure LDR potential divider pin as ADC input  buzzer=PWM(Pin(14)) #Configure buzzer pin for PWM output  score=0 #Start game with no points    def reset():  '''Reset score variable and provide user feedback before a new game'''  global score, servo1Value, servo2Value  servo1Value=5000  servo2Value=5000  score=0  beep(2)  print("Ready for new game")    def beep(numBeeps):  '''Turn the buzzer on and off a given number of times'''  print("Beep")  for i in range(numBeeps):  buzzer.duty\_u16(1000)  buzzer.freq(1000)  sleep(0.2)  buzzer.duty\_u16(0)  sleep(0.1)  def winningSound():  '''Play a nice winning sound'''  print("Winner!")  beep(3)  buzzer.duty\_u16(5000)  #add a better winning sound here - eg loop through a range of frequencies in quick succession    reset()  from random import randint  while True:  if LDR.read\_u16()<5000: #Strong light detected - change this value to suit your room and LDR  servo1Value=servo1Value+randint(-500,500)  servo2Value=servo2Value+randint(-500,500)  if servo1Value<3000:  servo1Value=3000  if servo1Value>7000:  servo1Value=7000  if servo2Value<3000:  servo2Value=3000  if servo2Value>7000:  servo2Value=7000  servo1.duty\_u16(servo1Value)  servo2.duty\_u16(servo2Value)  score=score+10  buzzer.freq(score+100)  buzzer.duty\_u16(score+1000)  print(score)  if score>1000: #If the game has been won, play a sound and get ready for the next round  winningSound()  reset()  else: #Strong light not detected .. do nothing (turn the buzzer off)  buzzer.duty\_u16(0)  sleep(0.1) |

## Explore the algorithmic patterns, code and applications for open and closed control systems

**Activity 36**: open and closed loop control system research

Watch [Open loop and closed loop systems](https://youtu.be/WXDfOTlv4xE) (3:57) and read [Open-loop and closed-loop](https://en.wikipedia.org/wiki/Control_loop#Open-loop_and_closed-loop).

1. Draw diagrams representing open loop and closed loop control.

|  |  |
| --- | --- |
| Open loop control | Closed loop control |
|  |  |

1. Describe how the error is determined in a closed loop control algorithm.

|  |
| --- |
| Sample answer:  Sensors measure the state of the system after control output and these values are used to find how far away from a desired set point the system is. The difference between the system state and the desired setpoint is the error. |

1. Explain an advantage and a disadvantage of closed loop control.

|  |
| --- |
| Sample answer:  Because closed loop control constantly monitors the result of its output, it can adjust its control values to achieve a system state closer to the desired set point. Closed loop control is more complex to achieve and can introduce instability into the system. |

1. Classify each of the following types of systems as open or closed loop.

|  |  |
| --- | --- |
| System type | Open or closed loop? |
| TV remote |  |
| Vehicle cruise control |  |
| Vehicle steering |  |
| Ceiling fan speed control |  |
| Air conditioner temperature control |  |
| Washing machine cycle timing |  |
| Traffic light control |  |
| A robotic arm programmed to perform a complex series of predetermined movements |  |
| A line following robot |  |

1. Recommend and justify the use of open or closed loop control for a programmable robotic arm that moves components from one place to another in a production line.

|  |
| --- |
| Sample answers could include:  Open loop control: Using quality stepper or servo motors, enough accuracy in position may be achieved in arm position.  The components on a production line would have an entirely predictable weight so the motors will maintain accurate positions under load.  Arm movements could be repeated exactly without measurement as the components source and destination positions wouldn’t change relative to the robotic arm.  Closed loop control: A production line needs very high reliability, so even though a system could be designed with open loop control, the production line may fail if there is any error in component positioning.  The robotic arm should continually sense the difference between its position and the position of the component to be moved so that it can actuate its gripper at exactly the right point. |

## Outline the features of an algorithm and program code used for autonomous control

**Activity 37**: autonomous control algorithm research

There are 3 common types of algorithms used for autonomous control of mechatronic systems. They all sense the system state and use feedback to calculate the error from a desired setpoint. They then use this error to compute the required control values before restarting the main control loop. These algorithms are known as:

* On/off (or ‘bang-bang’) control ([Wikipedia article](https://en.wikipedia.org/wiki/Bang%E2%80%93bang_control))
* Proportional control ([Wikipedia article](https://en.wikipedia.org/wiki/Proportional_control))
* PID (proportional, integral and derivative) control ([Wikipedia article](https://en.wikipedia.org/wiki/Proportional%E2%80%93integral%E2%80%93derivative_controller)).

Research each of these algorithms using the articles linked above to help answer the following questions.

1. Which of these algorithms would be considered closed loop algorithms?

|  |
| --- |
| Sample answer:  All 3 of these algorithms can be considered closed loop algorithms as they can all sense and feed back the system state to determine the error from a desired setpoint. As long as the measurements used to determine whether to turn actuators on/off are based on things the actuators can affect, then the system is considered closed loop. For example, a temperature controller turning a heating element on/off based on the measured temperature would be considered closed loop.  The on/off control algorithm would be considered open loop if the system isn’t measuring data from the environment that it is affecting through its own control or actuators. For example, if a temperature controller is being turned on/off from a simple timer, rather than from measuring the temperature, the system would be considered open loop. |

1. Why is on/off control sometimes referred to as ‘bang-bang’ control?

|  |
| --- |
| Sample answer:  The system changes until it hits a limit, then changes to go in the opposite direction until it hits the opposite limit. It keeps going back and forth between these limits. |

**Extension:** explain the purpose of hysteresis in on/off control algorithms.

|  |
| --- |
| Sample answer:  On/off control algorithms turn an actuator on or off, depending on the system state (for example whether it is below or above a desired set point). To avoid the system turning on and off very quickly (which may damage components), a tolerance for error is introduced so the system won’t turn on or off when it is within this tolerated error. This makes the system overshoot until it switches actuator state, then move in the opposite direction until it undershoots and switches the actuator state again. |

1. How does proportional control improve control of a system over on/off control?

|  |
| --- |
| Sample answer:  Proportional control sets the size of the response based on the difference from the desired setpoint. The bigger the difference, the bigger/faster the response. |

1. What is a common weakness of proportional control algorithms when the error is small?

|  |
| --- |
| Sample answer:  When the error between the system state and the desired setpoint is small, the control values sent to the actuators may be too small to actually move them in a real system. This means that the system may end up stabilising a small distance away from the desired setpoint. |

1. Describe how the integral and derivative control values in a PID controller improve on simple proportional control.

|  |
| --- |
| Sample answer:  The integral component adds the error over time so will eventually grow to a large value if proportional control stabilises the system away from the desired setpoint.  The derivative component uses the change in error to measure how fast the system is approaching the desired setpoint. If the system is moving rapidly away from the desired setpoint, the control values will be substantially adjusted to help move the system back towards the setpoint more quickly. If the system is moving rapidly toward the setpoint, the control values will be substantially adjusted in the opposite direction of the current movement to limit the chance of overshoot of the setpoint. |

# Programming and building – user-programmable articulating arm

**Teacher note:** this section is a guided activity to help students create a prototype of an articulated robotic arm with an attached gripper.

It builds upon the hardware from the previous light sensor game, adding a gripper, potentiometer and 3 buttons.

The gripper could be substituted for a servo if a suitable gripper is not available.

Suggested parts include:

* microcontroller board with connection to a power source
* jumper wires
* potentiometers x3
* buttons x3
* servo x2
* gripper with servo x1
* piezoelectric buzzer x1
* separate 5–6V power source if using larger servos, heavier gripper, longer arms or planning to load the servos in any way.

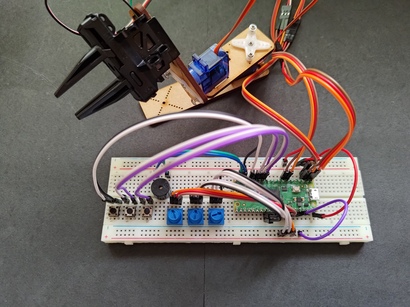
Extra support files available in the [Software Engineering channel of the TAS Statewide Staffroom](https://teams.microsoft.com/l/channel/19%3Ac4aa94ee6a3340e3b35ca2f2c40375df%40thread.tacv2/13.%20Software%20Engineering%2011-12?groupId=cd5a04e1-7742-47dd-b141-9519486d9e00&tenantId=05a0e69a-418a-47c1-9c25-9387261bf991) include the following:

* video showing a sample project
* link to a simulated project on [wokwi.com](https://wokwi.com/)
* Python code for running the sample project on a Raspberry Pi Pico
* Adobe Illustrator file for laser cutting servo support pieces using 3 mm material
* links showing alternative design ideas of similar articulated arms.

In this section, you will be building and coding an open loop user-programmable articulating robotic arm.

You could think of this as a prototype for a reprogrammable industrial-scale device in an automated production line designed to pick up and move components in a repeated fashion.

Figure 35 – sample complete project



As a class, watch [the 2 coolest robots in Tesla’s factory (1:24)](https://www.youtube.com/watch?v=WYnOGAvQEgk).

Discuss the common features of articulated robotic arms.

This project will reuse and build upon the interface from your previous sensor and servo game. You will add a gripper and an extra potentiometer to control it.

Potentiometers will be used to set positions of the arms and grippers for each recordable set point. You will also add some push buttons to allow an end user to record, delete and play back their own set points.

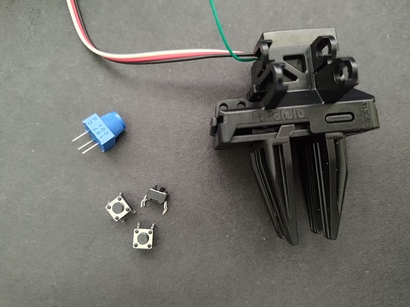
When the set points are played back, the articulated arm and gripper should move through the list of set points recorded by the user to emulate an industrial robotic arm.

## Design, develop and produce a mechatronic system for a real-world problem

Collect the following parts to add to the hardware from your previous light sensor game:

* potentiometer x1
* push buttons x3
* gripper with servo x1
* jumper wires

Figure 36 – extra parts required



Remove the light sensor from the previous game as it is not needed for this project.

### Software control

#### Potentiometer control

Read about connecting and testing potentiometers for [Arduino](https://docs.arduino.cc/built-in-examples/basics/AnalogReadSerial) or [Pico](https://projects.raspberrypi.org/en/projects/introduction-to-the-pico/11).

Connect the extra potentiometer using a pinout diagram for your microcontroller board.

Use a pinout diagram for your microcontroller board to ensure that you connect to appropriate pins. If you run out of available holes for a power or ground connection on your breadboard, you can connect either power or ground to an unused row on the breadboard and use this area for connecting.

Create a small program to test that you can read all 3 potentiometer values.

In a loop, read and print each potentiometer value.

Use the sample wiring diagrams with code for [Arduino](https://www.tinkercad.com/things/4BidqOuVugn-articulated-arm-potentiometer-control?sharecode=ceO_nUTLKh-Q55Pn01c0frk0Y5AaSIQj8Hv84kTA8tY) and [Pico](https://wokwi.com/projects/383423687620008961) if needed.

### Gripper control

Connect the gripper’s servo to your circuit using a pinout diagram for your microcontroller board. Consider the total power requirements of the servos as you may need to power them separately to your microcontroller board.

A 4x AA or 4x AAA battery pack can be used to provide power to micro servos.

If you use a battery pack, connect its ground to a ground on your microcontroller board to provide a common reference level for the PWM control signal.

Create a small program to test that you can control the gripper and the other 2 servos.

Use the sample wiring diagrams with code for [Arduino](https://www.tinkercad.com/things/hbJ1N0RiyBt-articulated-arm-servo-control?sharecode=Gv3a2W0XO6du-Gs29UeJUmamkQC1SnJQDvscaKrxfzo) and [Pico](https://wokwi.com/projects/383424942885241857) if needed.

Note that the sample wiring diagram doesn’t show a battery pack to power the servos, even though you may require one.

### Potentiometer control of servos

Combine your working code from the potentiometer and gripper control tests into a single program file to allow each potentiometer to control a servo. Your program should include the following features:

* Importing or including any required libraries
* Configuration of analog input pins from each potentiometer
* Configuration of pulse width modulated output pins for each servo
* A main loop that repeatedly does the following for all 3 potentiometer/servo pairs
* reads the potentiometer value
* scales the value suitable for use in your servo
* controls the servo using the scaled value
* adds a small delay of about 100ms before the next iteration.

Record an image of your working circuit and its code in the table below.

|  |  |
| --- | --- |
| Code to control servos with potentiometers | Image of circuit |
|  |  |

### Mechanical engineering

You may have the opportunity to create, cut, print or gather parts to connect the entire system together.

An articulated robotic arm must be connected with minimal flex and must remain stable in its operation so parts can be reliably located, gripped and moved as needed.

1. Describe mechanical considerations in the design or operation of your articulated robotic arm.

|  |
| --- |
| Sample answers may include:  Weight: if the servos are heavy, the supporting arms will need to be strong. The servos may also need to be bolted into position. This will all increase the weight further, requiring a separate power supply for the servos and possibly also stronger (and heavier) servos.  Arm length: the longer the arms are, the more load placed on the servos. This will require a separate servo power supply and possibly also stronger (and heavier) servos. Longer arms will also place more stress on the supporting arms which may need to be made stronger.  Balance: the base of the system will need to be held or weighted down to stop it falling over when stretched out, moving, or carrying a load.  Movement: moving components will need to be connected or coded so that they don’t hit or block other parts when moving. |

1. Describe how you plan to connect a gripper or servo to the articulated arm and keep the entire system physically stable while in operation.

|  |
| --- |
| Sample answers could include:  I will press-fit the gripper’s servo into a laser-cut hole in 3 mm MDF and screw in one of its ends.  I will hot-glue the gripper’s base to the end of the final articulated arm.  I will press-fit the base servo into a long base-plate that can be held in place under a heavy object such as a textbook or laptop. |

### Electronics and mathematics

#### Reduce potentiometer noise

You may have noticed that the value read from each potentiometer is not very stable.

Cheap potentiometer construction and unwanted noise in the circuit affects the readings from the potentiometer.

Software can be used to filter out some of this noise using an average (mean) value over a group of readings.

BEGIN mean(pin, numReadings)

REM return the mean of a number of readings from a given pin

total=0

FOR i=1 to numReadings STEP 1

total=total+analogRead(pin)

NEXT i

RETURN int(total/numReadings)

END

Create a function called *mean* to implement this logic in your code. Record the function in the space below.

|  |
| --- |
| Sample answers could include:  For Arduino/C++:  int mean(int pin, int numReadings){  //return the mean of a number of readings from a given pin  int total=0;  for(int i=0; i<numReadings; i=i+1){  total=total+analogRead(pin);  }  return int(total/numReadings);  }  For Pico/Python:  def mean(potentiometer, numReadings):  '''return the mean of a number of readings from a potentiometer'''  total=0  for i in range(numReadings):  total=total+potentiometer.read\_u16()    return int(total/numReadings) |

### Connect and test buttons

You will be using the 3 buttons to program the articulated arm and gripper.

You will be able to save positions that your robotic arm will move between.

This could then be implemented in a production line setting, with your system repeating the same action as many times as required.

The buttons will be used as follows:

* One button to add a new set point
* One button to remove a saved set point
* One button to move the robotic arm through each of the saved set points in turn.

Read [Digital Input Pull-up Resistor](https://docs.arduino.cc/tutorials/generic/digital-input-pullup) (Arduino/C++) or [Button with Pi Pico](https://highvoltages.co/tutorial/pi-pico/button-with-pi-pico/) (Pico/Python) to learn how to connect and use a button as a digital input to your microcontroller board.

1. Describe the purpose of a pull-up resistor when connecting a push button in a microcontroller circuit.

|  |
| --- |
| Sample answer:  If you connect a push button directly between Vcc and ground, you will create a short circuit between them when the button is being pushed. This will use a lot of current and may reset or damage the microcontroller.  If you add a resistor in the circuit as shown, the current will be limited to a safe value when the button is pressed.  When the button is open, D5 will be connected to Vout; when the button is closed, D5 will be connected directly to ground.  Using an internal pull-up resistor on D5 removes the need to supply and connect your own resistor, so the button can be connected directly between ground (or Vout) and an input.Model showing sample wiring between a microcontroller and a button using an external resistor. |

Physically connect 3 buttons to your circuit.

From each button there should be one connection to a digital input pin and the other to either ground or the board’s logic level.

Use the simulations on [Tinkercad](https://www.tinkercad.com/things/iPnuvPMsZUx-articulated-arm-button-test?sharecode=cULREvCNCN7WSJv3fX8REIG5FIQuO859zCS8piIzvXE) or [Wokwi](https://wokwi.com/projects/383432370217383937) to check your wiring.

Adapt the sample code provided to verify that you can read values correctly from all 3 buttons in your circuit.

Record your code to test and verify button functionality in your circuit below.

|  |
| --- |
| Sample answers could include:  For Arduino/C++:  void setup()  {  //Configure pushbuttons for digital input with pull-up resistors  pinMode(10,INPUT\_PULLUP);  pinMode(11,INPUT\_PULLUP);  pinMode(12,INPUT\_PULLUP);  Serial.begin(9600);  }  void loop(){  Serial.print(digitalRead(10));  Serial.print(" ");  Serial.print(digitalRead(11));  Serial.print(" ");  Serial.println(digitalRead(12));  delay(100);  }  For Pico/Python:  from machine import Pin  from time import sleep  #Configure the button pins for digital input with pull-up resistors  setButton=Pin(10,Pin.IN,Pin.PULL\_UP)  deleteButton=Pin(11,Pin.IN,Pin.PULL\_UP)  playButton=Pin(12,Pin.IN,Pin.PULL\_UP)    while True:  print(setButton.value(), deleteButton.value(), playButton.value())  sleep(0.1) |

## Implement algorithms and design programming code to drive mechatronic devices

When your microcontroller first connects to the servos it repositions them. This can require a large surge of electrical current which has the potential to damage the microcontroller or the device powering it.

To help reduce the risk of damage, you will add logic to reposition one servo at a time until all servos are able to be safely controlled by the potentiometers.

Create a reset() procedure that performs the actions listed below. Call this reset () procedure in your program’s setup phase, before the main loop begins:

* Set the position of servo1 to match the scaled value of potentiometer1
* Arduino/C++ example:   
   long int potentiometer1=analogRead(A1); servo1.write(int(potentiometer1\*180/1024));
* Pico/Python example: servo1.duty\_u16(int(3000+potentiometer1.read\_u16()\*4000/65536))
* Pause execution for one second to allow the servo to reach the desired position
* Arduino/C++ example: delay(1000);
* Pico/Python example: sleep(1)
* Set the position of servo2 to match the scaled value of potentiometer2
* Pause execution for one second to allow the servo to reach the desired position
* Set the position of servo3 to match the scaled value of potentiometer3
* Pause execution for one second to allow the servo to reach the desired position
* Print a status message as debugging output.

Incomplete simulations have been provided in [Tinkercad](https://www.tinkercad.com/things/78I0AeUd9H2-articulated-arm-for-student-completion?sharecode=eM9nyRDK_4UWNGyryvvQoqT0rRl-LEqmSmibSd8yULQ) and [Wokwi](https://wokwi.com/projects/383435937768138753) to test your reset() procedure. You should copy and save these into your own account so you can continue to use any changes that you make. Once your reset() procedure is working, upload it to your microcontroller board. Record your procedure’s code below.

|  |
| --- |
| Sample answers could include:  For Arduino/C++:  void reset(){  //Adjust servos one at a time to limit power draw  long int potentiometer1=analogRead(A1);  servo1.write(int(potentiometer1\*180/1024));  delay(1000);    long int potentiometer2=analogRead(A2);  servo2.write(int(potentiometer2\*180/1024));  delay(1000);    long int potentiometer3=analogRead(A3);  servo3.write(int(potentiometer3\*180/1024));  delay(1000);    Serial.println("Servos reset and ready to follow potentiometers");  }  For Pico/Python:  def reset():  '''Adjust servos one at a time to limit power draw'''  servo1.duty\_u16(int(3000+potentiometer1.read\_u16()\*4000/65536))  sleep(1)  servo2.duty\_u16(int(3000+potentiometer2.read\_u16()\*4000/65536))  sleep(1)  servo3.duty\_u16(int(3000+potentiometer3.read\_u16()\*4000/65536))  sleep(1)  print("Servos reset and ready to follow potentiometers") |

## Develop simulations and prototypes of a potential mechatronic system to test programming code

Some online programming environments and simulators are linked below.

* <https://maker.makecode.com/>
* <https://makecode.microbit.org/>
* <https://create.arduino.cc/editor>
* <https://www.tinkercad.com/circuits>
* <https://wokwi.com/>

Create or adapt a simulation to match your physical circuit if you have not already done so. All wiring connections should be the same on your physical circuit.

This will allow you to copy and paste code between the simulated and physical circuits easily.

Record the URL of your own simulation below.

|  |
| --- |
| Simulation URL |
|  |

**Teacher note**: the syllabus section ‘Design, develop and implement programming code for a closed loop control system’ is not covered in this project as it uses an open loop control algorithm. Closed loop control is covered in the [self-stabilising ball project](#_Programming_and_building).

## Apply programming code to integrate sensors, actuators and end effectors/manipulators

To make your robotic arm programmable, you will need to add functionality to the buttons.

When controlling the robotic arm with the potentiometers, you will be able to press a button to record the current position as a setpoint.

You will also be able to press a button to remove a setpoint or to make the robot iterate through all of the saved setpoints.

Identify and describe an appropriate data structure to store multiple setpoints for your robotic arm.

|  |
| --- |
| Sample answer:  Since there will be multiple setpoints that we will need to iterate through, we could use a list or an array.  Each item will need to record a value for each of the 3 servos (either the value of the corresponding potentiometer or the value to send to the corresponding servo).  These could be stored as a record with 3 integer variables. |

Follow the instructions below in your simulated circuit.

When you can successfully add, delete and play setpoints in your simulation, test your code in a physical circuit.

**Step 1:** create a data structure to store each setpoint and a variable to hold a number of these setpoints.

Record the code that defines your data structure and initial empty variable below.

|  |
| --- |
| Sample answers could include:  For Arduino/C++:  struct Setpoint{  int servo1, servo2, servo3;  };  Setpoint setpoints[100];  int setpointCount=0;  For Pico/Python:  setpoints=[]  class Setpoint:  servo1=None  servo2=None  servo3=None |

**Step 2:** change the logic in your main loop to have a structure similar to the following:

IF setButton pressed THEN

PRINT("Adding setpoint")

ELSE IF deleteButton pressed THEN

PRINT("Deleting a setpoint")

ELSE IF playButton pressed THEN

PRINT("Playing setpoints")

ELSE

REM Control servos using potentiometers

END IF

SLEEP for 100ms

Record the code from your main loop below.

|  |
| --- |
| Sample answers could include:  For Arduino/C++:  void loop(){  long int potentiometer1=analogRead(A1);  long int potentiometer2=analogRead(A2);  long int potentiometer3=analogRead(A3);  if(digitalRead(10)==0){ //add a new setpoint  Serial.println("Adding setpoint");  }else if(digitalRead(11)==0){ //delete the last setpoint  Serial.println("Deleting a setpoint");  }else if(digitalRead(12)==0){ //play the recorded setpoints  Serial.println("Playing setpoints");  }else{ //control servos with the potentiometers  servo1.write(int(potentiometer1\*180/1024));  servo2.write(int(potentiometer2\*180/1024));  servo3.write(int(potentiometer3\*180/1024));  }  delay(100);  }  For Pico/Python:  while True:  if setButton.value()==0: #add a new setpoint  print("Adding setpoint")  elif deleteButton.value()==0: #delete the last setpoint  print("Delete a setpoint")  elif playButton.value()==0: #play the recorded setpoints  print("Play setpoints")  else: #control servos with the potentiometers  servo1.duty\_u16(int(3000+potentiometer1.read\_u16()\*4000/65536))  servo2.duty\_u16(int(3000+potentiometer2.read\_u16()\*4000/65536))  servo3.duty\_u16(int(3000+potentiometer3.read\_u16()\*4000/65536))  sleep(0.1) |

**Step 3:** create code to add setpoints when the set button is pressed.

In a new setpoint, store the 3 values read from the potentiometers into a record or similar data structure.

Add this data to the variable that stores all setpoints.

Record your code to add a setpoint below.

|  |
| --- |
| Sample answers could include:  For Arduino/C++:  if(analogRead(10)==0){ //add a new setpoint  if(setpointCount<100){  setpoints[setpointCount]=potentiometerPoint();  setpointCount=setpointCount+1;  }  }  For Pico/Python:  if setButton.value()==0: #add a new setpoint  setpoints.append(potentiometerPoint()) |

**Step 4:** create code to remove or inactivate setpoints when the delete button is pressed. Check if there are any stored setpoints before deleting the last setpoint.

Record your code to delete a setpoint below.

|  |
| --- |
| Sample answers could include:  For Arduino/C++:  }else if(analogRead(11)==0){ //delete the last setpoint  if(setpointCount>0){  setpointCount=setpointCount-1; //WARNING: inactivated only  }  }  For Pico/Python:  elif deleteButton.value()==0: #delete the last setpoint  if len(setpoints)>0:  setpoints=setpoints[0:len(setpoints)-1] |

**Step 5:** create code to iterate the robotic arm through setpoints.

Access each setpoint in a loop and send an appropriately scaled control signal to each servo followed by a delay of about 500ms.

The delay will help prevent power surges from damaging connected equipment.

In the space below write code that it iterates through the setpoints.

|  |
| --- |
| Sample answers could include:  For Arduino/C++:  }else if(digitalRead(12)==0){ //play the recorded setpoints  for(int i=0; i<setpointCount; i=i+1){  servo1.write(setpoints[i].servo1);  delay(500);  servo2.write(setpoints[i].servo2);  delay(500);  servo3.write(setpoints[i].servo3);  delay(1000);  }  For Pico/Python:  elif playButton.value()==0: #play the recorded setpoints  for setpoint in setpoints:  servo1.duty\_u16(setpoint.servo1)  sleep(0.5)  servo2.duty\_u16(setpoint.servo2)  sleep(0.5)  servo3.duty\_u16(setpoint.servo3)  sleep(1) |

**Step 6:** apply this code to your physical system and test it.

## Implement specific control algorithms that enhance the performance of a mechatronic system

**Teacher note**: Tinkercad can’t run some of the code in this section. If using Arduino/C++, code this section directly in your IDE instead of using Tinkercad.

At this point, the movements of your robotic arm are not very smooth.

In this section, you will design and implement algorithms to control the movement of the arm smoothly between setpoints.

This requires interpolating between adjacent setpoints in a loop.

Between each of the setpoints, you will make the system move between many intermediate points to control the motion more smoothly.

Follow the steps described below to create smooth movement between all setpoints.

**Step 1:** use the pseudocode below to create a procedure that moves all servos in a smooth motion between a given start and an end setpoint.

Note that the Tinkercad simulator doesn’t pass parameters. Incorporate the code in the main program loop without using a subprogram.

BEGIN move(start, end)

FOR i=1 to 100 STEP 1

SEND a control signal to servo1 of   
 start.servo1 + (end.servo1-start.servo1)\*i/100

SEND a control signal to servo1 of   
 start.servo2 + (end.servo2-start.servo2)\*i/100

SEND a control signal to servo1 of   
 start.servo3 + (end.servo3-start.servo3)\*i/100

SLEEP for 10ms

NEXT i

END

Record your procedure to move smoothly between a start and end setpoint below.

|  |
| --- |
| Sample answers could include:  For Arduino/C++:  void move(Setpoint start, Setpoint end){  //Slowly move the servos from a start to an end point  int numSteps=100, milliseconds=1000;  for(int i=0; i<100; i++){  servo1.write(int(start.servo1+(end.servo1-start.servo1)\*i/numSteps));  servo2.write(int(start.servo2+(end.servo2-start.servo2)\*i/numSteps));  servo3.write(int(start.servo3+(end.servo3-start.servo3)\*i/numSteps));  delay(millseconds/numSteps);  }  }  For Pico/Python:  def move(start,end):  '''Slowly move the servos from a start to an end point'''  numSteps=100  seconds=1  for i in range(numSteps):  servo1.duty\_u16(int(start.servo1+(end.servo1-start.servo1)\*i/numSteps))  servo2.duty\_u16(int(start.servo2+(end.servo2-start.servo2)\*i/numSteps))  servo3.duty\_u16(int(start.servo3+(end.servo3-start.servo3)\*i/numSteps))  sleep(seconds/numSteps) |

**Step 2:** create a function to return a setpoint defined by the current position of the potentiometers. The function should read and scale each potentiometer value appropriate for servo control.

Each scaled value should then be assigned to a field in your setpoint data structure.

Incomplete sample functions are given within the box below to adapt as required by your system.

|  |
| --- |
| Sample incomplete **Pico/Python** function:  def potentiometerPoint():  '''Returns current potentiometer point'''  point=Setpoint()  point.servo1=int(3000+potentiometer1.read\_u16()\*4000/65536)  #add code here to store values for servo2 and servo3  return point  Sample **Arduino/C++** function:  Setpoint potentiometerPoint(){  //Returns current potentiometer point  Setpoint point;  long int potentiometer1=analogRead(A1);  point.servo1=int(potentiometer1\*180/1024);  //add code here to store values for servo2 and servo3  return point;  } |

Record your procedure to move smoothly between a start and end setpoint below.

|  |
| --- |
| Sample answers could include:  For Arduino/C++:  struct Setpoint potentiometerPoint(){  //Returns current potentiometer point  Setpoint point;  long int potentiometer1=analogRead(A1);  point.servo1=int(potentiometer1\*180/1024);  long int potentiometer2=analogRead(A2);  point.servo2=int(potentiometer2\*180/1024);  long int potentiometer3=analogRead(A3);  point.servo3=int(potentiometer3\*180/1024);  return point;  }  For Pico/Python:  def potentiometerPoint():  '''Returns current potentiometer point'''  point=Setpoint()  point.servo1=int(3000+potentiometer1.read\_u16()\*4000/65536)  point.servo2=int(3000+potentiometer2.read\_u16()\*4000/65536)  point.servo3=int(3000+potentiometer3.read\_u16()\*4000/65536)  return point |

**Step 3:** adjust the code in your main loop to make smooth movements.

Locate the section in your code where the playButton has been pressed.

Use the provided sample flowchart or pseudocode to edit the code in your main loop or create a subprogram to handle this logic.

The sample assumes the setpoints are stored in an array of records.

Adjust the code for your data structure.

|  |  |
| --- | --- |
| Sample flowchart for smooth movement | Sample pseudocode for smooth movement |
| Sample portion of flowchart showing smooth movement code design. | IF len(setpoints)>0 THEN  start=potentiometerPoint()  FOR i=0 to len(setpoints) STEP 1  move(start, setpoints[i])  start=setpoints[i]  NEXT i  move(start, potentiometerPoint())  END IF |

Record your code to move smoothly between all setpoints below.

|  |
| --- |
| Sample answers could include:  For Arduino/C++:  }else if(digitalRead(12)==0){ //play the recorded setpoints  if(setpointCount>0){  Setpoint start=potentiometerPoint();  for(int i=0; i<setpointCount; i=i+1){  move(start, setpoints[i]);  start=setpoints[i];  }  move(start, potentiometerPoint());  }  }  For Pico/Python:  elif playButton.value()==0: #play the recorded setpoints  if len(setpoints)>0:  start=potentiometerPoint()  for setpoint in setpoints:  move(start,setpoint)  start=setpoint  beep(1)  move(start,potentiometerPoint()) |

## Design, develop and implement a user interface (UI) to control a mechatronic system

While testing the development of code for your articulated arm, you probably used debug output statements to view the status of the system.

It is also important for users of your system to have feedback.

Feedback can let users know when the system has received their inputs correctly and can also give some system status information.

Adding feedback to the user interface helps users interact with the system correctly.

****Brainstorm several different ways a developer could display the state of a mechatronic system or output some debugging information when there is no screen. Record your ideas below.

|  |
| --- |
| **Sample answer:**  Sounds and lights to show codes or status information, or an LCD display to show readable text. |

Using a buzzer and the code you created in the sensor and servo game, create and use a beep subprogram to use as user feedback.

Add beeps when the system is ready for use and for each of the push buttons.

When the system can’t respond to a user action, add 2 beeps to let the user know. This could be when there are no setpoints to play or delete.

Experiment by adding a beep at each setpoint during playback to see whether it improves the user experience.

Copy your final, complete code below.

|  |
| --- |
| Sample answers could include:  For Arduino/C++ ([Wokwi simulation](https://wokwi.com/projects/383461457677997057)):  //Articulated arm  #include <Servo.h>  Servo servo1, servo2, servo3;  struct Setpoint{  int servo1, servo2, servo3;  };  Setpoint setpoints[100];  int setpointCount=0;  void move(Setpoint start, Setpoint end){  //Slowly move the servos from a start to an end point  int numSteps=100, milliseconds=1000;  for(int i=0; i<100; i++){  servo1.write(int(start.servo1+(end.servo1-start.servo1)\*i/numSteps));  servo2.write(int(start.servo2+(end.servo2-start.servo2)\*i/numSteps));  servo3.write(int(start.servo3+(end.servo3-start.servo3)\*i/numSteps));  delay(milliseconds/numSteps);  }  }  struct Setpoint potentiometerPoint(){  //Returns current potentiometer point  Setpoint point;  long int potentiometer1=analogRead(A1);  point.servo1=int(potentiometer1\*180/1024);  long int potentiometer2=analogRead(A2);  point.servo2=int(potentiometer2\*180/1024);  long int potentiometer3=analogRead(A3);  point.servo3=int(potentiometer3\*180/1024);  return point;  }  void setup()  {  //Configure the potentiometer pins for ADC input  pinMode(A1,INPUT);  pinMode(A2,INPUT);  pinMode(A3,INPUT);    //attach each servo to a PWM-capable pin  //the servo library sets the 50Hz pulse frequency  servo1.attach(5, 1000, 2000);  servo2.attach(6, 1000, 2000);  servo3.attach(4, 1000, 2000);    //Configure pushbuttons for digital input with pull-up resistors  pinMode(10,INPUT\_PULLUP);  pinMode(11,INPUT\_PULLUP);  pinMode(12,INPUT\_PULLUP);    Serial.begin(9600);    reset();  }  void loop(){  long int potentiometer1=analogRead(A1);  long int potentiometer2=analogRead(A2);  long int potentiometer3=analogRead(A3);  if(digitalRead(10)==0){ //add a new setpoint  if(setpointCount<100){  beep(1);  setpoints[setpointCount]=potentiometerPoint();  setpointCount=setpointCount+1;  } else beep(2); //no more set points available  }else if(digitalRead(11)==0){ //delete the last setpoint  if(setpointCount>0){  beep(1);  setpointCount=setpointCount-1;  } else beep(2); //no setpoints to delete  }else if(digitalRead(12)==0){ //play the recorded setpoints  if(setpointCount>0){  beep(1);  Setpoint start=potentiometerPoint();  for(int i=0; i<setpointCount; i=i+1){  move(start, setpoints[i]);  start=setpoints[i];  delay(500);  }  move(start, potentiometerPoint());  beep(2); //finished playing back setpoints  } else beep(2); //no setpoints to play back  }else{ //control servos with the potentiometers  servo1.write(int(potentiometer1\*180/1024));  servo2.write(int(potentiometer2\*180/1024));  servo3.write(int(potentiometer3\*180/1024));  }    delay(100);  }  void reset(){  //Adjust servos one at a time to limit power draw  long int potentiometer1=analogRead(A1);  servo1.write(int(potentiometer1\*180/1024));  delay(500);  long int potentiometer2=analogRead(A2);  servo2.write(int(potentiometer2\*180/1024));  delay(500);  long int potentiometer3=analogRead(A3);  servo3.write(int(potentiometer3\*180/1024));  delay(500);  beep(2); //system ready for use  }  void beep(int numBeeps){  //Turn the buzzer on and off a given number of times  for(int i=0; i<numBeeps; i=i+1){  tone(9,1000);  delay(300);  noTone(9);  delay(200);  }  }  For Pico/Python ([Wokwi simulation](https://wokwi.com/projects/383437121741770753)):  #Articulated arm  from machine import Pin,ADC,PWM  from time import sleep  #Configure the servo pins for PWM output  servo1=PWM(Pin(0))  servo2=PWM(Pin(1))  servo3=PWM(Pin(5))  servo1.freq(50)  servo2.freq(50)  servo3.freq(50)  #Configure the potentiometer pins for ADC input  potentiometer1=ADC(Pin(26))  potentiometer2=ADC(Pin(27))  potentiometer3=ADC(Pin(28))  #Configure the button pins for digital input with pull-up resistors  setButton=Pin(10,Pin.IN,Pin.PULL\_UP)  deleteButton=Pin(11,Pin.IN,Pin.PULL\_UP)  playButton=Pin(12,Pin.IN,Pin.PULL\_UP)  buzzer=PWM(Pin(14)) #Configure buzzer pin for PWM output    def reset():  '''Adjust servos one at a time to limit power draw'''  servo1.duty\_u16(int(3000+potentiometer1.read\_u16()\*4000/65536))  sleep(1)  servo2.duty\_u16(int(3000+potentiometer2.read\_u16()\*4000/65536))  sleep(1)  servo3.duty\_u16(int(3000+potentiometer3.read\_u16()\*4000/65536))  sleep(1)  beep(2) #system ready for use  def beep(numBeeps):  '''Make the buzzer beep a given number of times'''  buzzer.freq(1000) #Set the frequency  for i in range(numBeeps):  buzzer.duty\_u16(1000)  sleep(0.2)  buzzer.duty\_u16(0)  sleep(0.1)  def move(start,end,numSteps=100,seconds=1):  '''Slowly move the servos from a start to an end point'''  for i in range(numSteps):  servo1.duty\_u16(int(start.servo1+(end.servo1-start.servo1)\*i/numSteps))  servo2.duty\_u16(int(start.servo2+(end.servo2-start.servo2)\*i/numSteps))  servo3.duty\_u16(int(start.servo3+(end.servo3-start.servo3)\*i/numSteps))  sleep(seconds/numSteps)  def potentiometerPoint():  '''Returns current potentiometer point'''  point=Setpoint()  point.servo1=int(3000+potentiometer1.read\_u16()\*4000/65536)  point.servo2=int(3000+potentiometer2.read\_u16()\*4000/65536)  point.servo3=int(3000+potentiometer3.read\_u16()\*4000/65536)  return point  reset()  setpoints=[]  class Setpoint:  servo1=None  servo2=None  servo3=None  while True:  if setButton.value()==0: #add a new setpoint  setpoints.append(potentiometerPoint())  beep(1)  elif deleteButton.value()==0: #delete the last setpoint  if len(setpoints)>0:  setpoints=setpoints[0:len(setpoints)-1]  beep(1)  else:  beep(2)  elif playButton.value()==0: #play the recorded setpoints  if len(setpoints)>0:  beep(1)  start=potentiometerPoint()  for setpoint in setpoints:  move(start,setpoint)  start=setpoint  beep(1)  move(start,potentiometerPoint())  beep(2) #setpoint playback complete  else:  beep(2) #no setpoints to play  else: #control the servos with the potentiometers  #these lines set the duty cycle to a value between 3000 & 7000  #which corresponds to a pulse width of 1ms to 2ms  servo1.duty\_u16(int(3000+potentiometer1.read\_u16()\*4000/65536))  servo2.duty\_u16(int(3000+potentiometer2.read\_u16()\*4000/65536))  servo3.duty\_u16(int(3000+potentiometer3.read\_u16()\*4000/65536))    sleep(0.1) |

It would not be appropriate to use a breadboard as part of the user interface for a commercial product.

Sketch a design for a better physical interface that could house the buttons and potentiometers on the outside with enough space on the inside to house the required electronic components.

Ensure you consider how your device will be powered in your design.

Design issues to consider include:

* physical robustness: the device should be able to withstand handling by untrained users
* simple and intuitive interface: components that work together should be located together; labels or diagrams should indicate the purpose of different components; the user should know whether the system is powered on.

**Extension**: construct a physical interface.

|  |
| --- |
| Sample answer:  Electronics, buzzer and battery housed inside. Battery has removable cover for replacement. Power button is lit when on.  Sample physical interface with potentiometers and buttons mounted on a box. |

## Create and use unit tests to determine the effectiveness and repeatability of each component’s control algorithm

Using the simulators **Arduino/C++** ([Wokwi simulation](https://wokwi.com/projects/383461457677997057)) and **Pico/Python** ([Wokwi simulation](https://wokwi.com/projects/383437121741770753)), students will be able to replicate the behaviour of the hardware without the need for physical components. Manual testing may be necessary, especially for aspects that are challenging to simulate or for hardware interactions that depend on the physical properties of the components.

Unit testing is where individual units or components of a program are tested in isolation. The goal is to verify that each unit of the software performs as designed.

A ‘unit’ refers to the smallest testable part of a program, typically a function or method.

Unit testing is crucial in software engineering because it:

* allows developers to catch and fix bugs early in the development process
* helps ensure code quality
* provides a safety net when making changes to the code.

Steps to create a unit test:

1. Isolate components: unit tests focus on testing individual components (functions, methods or classes) in isolation. This means that each unit test should only be concerned with the behaviour of one specific part of the code.
2. Write test cases: for each component, write test cases that check whether the component behaves as expected under different conditions.
3. These test cases typically include input values, the expected output, and any other relevant conditions.
4. Run tests: A testing framework (for example, [unittest](https://docs.python.org/3/library/unittest.html) in Python, or [ArduinoUnit](https://www.arduino.cc/reference/en/libraries/arduinounit/) for Arduino) can be used to automate the process of running the tests. The testing framework executes each test case and reports whether it passes or fails.
5. Include assertions: inside each test case include assertions (statements that check whether a given condition is true at a particular point in the code). These verify that the actual output of the component matches the expected output. If the assertions fail, it indicates that there might be a bug or unintended behaviour in the code.

Unit tests should be repeatable, meaning they should produce the same results every time they are run. Automating unit tests allows for frequent and consistent testing during development.

### Unit testing – samples

Study the example below in Python that uses the built-in unittest module.

Predict, Run, Investigate, Modify and Make other examples.

In this code the add function is being tested with 2 test cases.

The unittest framework executes the tests and checks whether the actual results match the expected results.

import unittest

def add(a, b):

return a + b

class TestAddFunction(unittest.TestCase):

def test\_add\_positive\_numbers(self):

result = add(2, 3)

self.assertEqual(result, 5)

def test\_add\_negative\_numbers(self):

result = add(-2, -3)

self.assertEqual(result, -5)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

To use unit tests for the Arduino code above, break down the code into testable units and then write tests for each unit.

The Arduino platform does not have built-in support for unit testing. A testing framework like [ArduinoUnit](https://www.arduino.cc/reference/en/libraries/arduinounit/) can be used or students could create their own testing mechanism.

A simplified example of how to structure the above code for testing:

#include <ArduinoUnit.h>

// Include the original code here

// Define a test fixture

test(ArmControlTests) {

// Test the move function

Setpoint start = {0, 0, 0};

Setpoint end = {90, 90, 90};

move(start, end);

// Add assertions to check if servos moved correctly

// Test the potentiometerPoint function

Setpoint point = potentiometerPoint();

// Add assertions to check if the potentiometer values are within expected range

// Add more tests for other functions if needed

}

// Run the tests

void setup() {

Serial.begin(9600);

runAllTests();

}

void loop() {

// Empty, as we only want to run tests in setup

}

To perform unit testing in Python, you could use a testing framework such as [unittest](https://docs.python.org/3/library/unittest.html). An example of how to structure the code for testing the control algorithm and using a simple test case for the move function is below:

import unittest

from unittest.mock import Mock

from your\_module import move, Setpoint, potentiometerPoint, reset, beep

class TestArmControl(unittest.TestCase):

def setUp(self):

# Set up mock objects or any necessary initialization

self.servo1 = Mock()

self.servo2 = Mock()

self.servo3 = Mock()

self.pot1 = Mock()

self.pot2 = Mock()

self.pot3 = Mock()

def test\_move\_function(self):

# Arrange

start = Setpoint(0, 0, 0)

end = Setpoint(90, 90, 90)

# Act

move(self.servo1, self.servo2, self.servo3, start, end, numSteps=10, seconds=0.1)

# Assert

self.servo1.duty\_u16.assert\_called()

self.servo2.duty\_u16.assert\_called()

self.servo3.duty\_u16.assert\_called()

# Add more test cases for other functions as needed

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

# Programming and building – self-stabilising ball on moving beam

**Teacher note:** this section is a guided activity to help students create a closed loop system to balance a ball on a rotating beam.

It uses the same interface as the previous programmable articulated arm, using a single servo and adding a distance sensor.

Suggested parts include:

* microcontroller board with connection to a power source
* jumper wires
* potentiometers x3
* buttons x3
* distance sensor (for example ultrasonic distance sensor or time-of-flight/lidar sensor) x1
* light emitting diode (LED) x1 with matching resistor (for example ~100-330 ohm) x1
* separate 5–6V power source if using larger servo, or planning to load the servo in any way.

Extra support files available in the [Software Engineering channel of the TAS Statewide Staffroom](https://teams.microsoft.com/l/channel/19%3Ac4aa94ee6a3340e3b35ca2f2c40375df%40thread.tacv2/13.%20Software%20Engineering%2011-12?groupId=cd5a04e1-7742-47dd-b141-9519486d9e00&tenantId=05a0e69a-418a-47c1-9c25-9387261bf991) include the following:

* video showing a sample project
* link to a simulated project on [wokwi.com](https://wokwi.com/)
* Python code for running the sample project on a Raspberry Pi Pico
* Adobe Illustrator file for laser cutting pieces using 3 mm material.

In this section, you will be building and coding a closed loop autonomous system designed to balance a ball on a moving beam. The beam will rotate to move a ball in either direction along a track while a distance sensor continually monitors the distance to the ball. The algorithm to control the angle of the beam will be a proportional–integral–derivative controller (PID controller).

As a class, watch a short video showing a [balancing ball on beam (0:53)](https://www.youtube.com/watch?v=tUDcpdTnhn4). Discuss how to construct a rotating beam, the electronic components required and the variables or code required to measure the ball position and control the beam angle.

This project will use the same user interface as your previous articulated arm project, although it will only require a single servo. The potentiometers will be used to adjust the values of 3 different variables used to control the system and the buttons will be used to set the desired position of the ball.

## Design, develop and produce a mechatronic system for a real-world problem

### Software control

As a class, watch [What is PID Control? (11:41)](https://www.mathworks.com/videos/understanding-pid-control-part-1-what-is-pid-control--1527089264373.html) and discuss how PID control could be used to balance a ball on a moving beam.

****Think: Have you ever noticed what happens to an elevator’s speed and position as it gets close to its destination floor? Does it slow down as it approaches its final position?

Does it sometimes end up a small distance from its destination? Does it sometimes overshoot the target and then come back to a more accurate position before opening the doors? What features of elevator control suggest it might be using PID control (or a subset of PID control)?

1. Describe how the proportional term in a PID controller is calculated and explain its purpose.

|  |
| --- |
| Sample answer:  The proportional term is how far away the system is from its target or set point. This is called the error and can be multiplied by a proportional constant for better control. In the ball and beam system, the proportional term could be how far the ball is from the centre. The further the ball is from the set point, the larger the proportional term and therefore the greater the control signal sent to the servo to move the beam. |

1. Describe how the integral term in a PID controller is calculated and explain its purpose.

|  |
| --- |
| Sample answer:  The integral term is the summed total over time of the error. When the ball is close to the set point, the error or proportional control will be too small to move it. The integral term will grow over time and will help reduce the steady state error. |

1. Describe how the derivative term in a PID controller is calculated and explain its purpose.

|  |
| --- |
| Sample answer:  The derivative term is the difference in the error between the current and previous readings. It lets us know how fast the ball is moving toward or away from the set point so that the control to the servo can be changed before the ball overshoots its target. |

**Extension**: describe how a PID controller can be tuned in a mechatronic system.

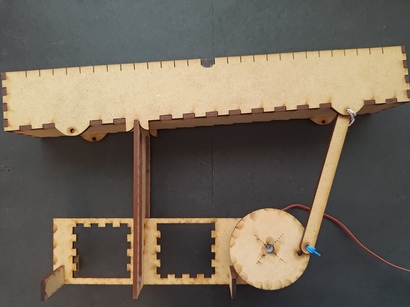
|  |
| --- |
| Sample answer:  Each PID component has its own constant which should be initially set to 0. Increase the constant of the proportional component until the system responds fast enough without too much overshoot, too slow a response or too much steady state error. Then, slowly increase the integral constant until the system reacts to steady state errors without introducing too much oscillation or instability. Then, increase the derivative constant until the system reacts more quickly to fast state changes without introducing instability. |

### Mechanical engineering

You will need to create a structure that has:

* a rotatable beam or arm which can
* support a ball evenly along its length
* hold your distance sensor in place at one end. If using an ultrasonic distance sensor, the beam needs to be able to mostly contain the sound waves reflected from the ball
* rotate up to 20 degrees either side of horizontal without physical constraints
* a support component that:
* will be reasonably stable while supporting the beam
* can help rotate the beam vertically either side of horizontal through the use of a servo or motor
* allows your microcontroller circuit and power supply to be connected nearby.

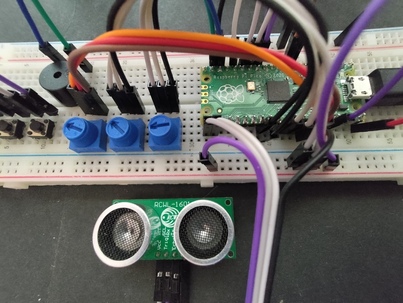
Figure 37 – sample supporting structure

* 

### Electronics and mathematics

In this step, you will connect and test all electrical components and apply initial scaling factors for the PID control algorithm. The user interface hardware is the same as for your previous programmable articulated arm project so can be reused. Create a new simulated project to test your code before applying it to the physical system.

Figure 38 – sample circuit



#### Servos

This project only requires a single servo, so 2 of the previous 3 can be removed. Add code to initialise a project with required libraries and to connect the single remaining servo. See the examples below.

For **Arduino/C++**:

//Balance beam - experiment and adjust to suit your circuit

#include <Servo.h>

Servo servo;

void setup()

{

//the servo library sets the 50Hz pulse frequency

servo.attach(7, 1000, 2000);

Serial.begin(9600);

}

For **Pico/Python**:

#Balance beam - experiment and adjust to suit your circuit

#use microsecond time measurements

from machine import Pin,ADC,PWM, time\_pulse\_us

from time import sleep, sleep\_us, ticks\_us

#Configure a servo pin for PWM output

servo=PWM(Pin(6))

servo.freq(50)

#### Distance sensor

Attach and test a distance sensor to your circuit.

Many distance sensors require the programmer to send a short pulse and then measure the time taken to receive the echo.

This can be done without any library but it is a good idea to create a function for this process.

See the examples below if needed.

For **Arduino/C++**:

//Extra global variables:

int trigger=2, echo=3;

//Extra setup() commands:

//configure pins for the ultrasonic distance sensor

pinMode(trigger,OUTPUT);

pinMode(echo,INPUT);

int singleReading() {

//return a single read from a sensor as a proxy for distance

digitalWrite(trigger, HIGH);

delayMicroseconds(5);

digitalWrite(trigger, LOW);

//measure the response time

int duration = pulseIn(echo, HIGH);

return duration;

}

For **Pico/Python**:

#Configure pins for the ultrasonic distance sensor

trigger=Pin(16,Pin.OUT)

echo=Pin(17,Pin.IN)

def singleReading(timeOut=5000):

'''return a single read from sensor as a proxy for distance'''

startTime=ticks\_us()

trigger.on()

sleep\_us(5)

trigger.off()

while echo.value()==0 and ticks\_us()-startTime<timeOut:

pass

startTime=ticks\_us()

while echo.value()==1 and ticks\_us()-startTime<timeOut:

pass

if ticks\_us()-startTime>timeOut:

return -1 #bad read - took too long

else:

return min(ticks\_us()-startTime,2000)

Test your distance sensor by calling the singleReading() function and printing its result repeatedly. Note the return value when the ball is in the centre of the beam for later use.

It may also be instructive to note how reliable the reading is at different distances from the sensor. See the examples below.

For **Arduino/C++**:

void loop(){ //main control loop

int loopStart=micros();

int reading=singleReading();

Serial.print(" reading="); Serial.println(reading);

}

For **Pico/Python**:

#main control loop

while True:

loopStart=ticks\_us()

reading=singleReading()

print("reading=%2d" % (reading))

#### Potentiometers and PID values

You will use the potentiometers for the proportional, integral, and derivative scaling factors in this project. This will allow you to adjust the values during program operation without having to reconnect, copy or recompile your code.

The code below configures 3 pins for analog input and scales their values to different ranges for initial PID control. Test that all potentiometers successfully adjust the PID scaling factors.

You may need to change this scaling for your system based on the range of values returned from your microcontroller’s ADC and the range of values needed to control your servos.

For **Arduino/C++**:

//Extra global variables:

int potentiometerP=A1, potentiometerI=A2, potentiometerD=A3;

//Extra setup() configuration:

//Configure the potentiometer pins for ADC input

pinMode(potentiometerP,INPUT);

pinMode(potentiometerI,INPUT);

pinMode(potentiometerD,INPUT);

//Extra main control loop code:

//get and scale values from potentiometers

float P=analogRead(potentiometerP)/1024.0/10.0; //0-0.1

float I=analogRead(potentiometerI)/1024.0/1000.0; //0-0.001

float D=analogRead(potentiometerD)/1024.0; //0-1

int reading=singleReading();

Serial.print(" P="); Serial.print(P);

Serial.print(" I="); Serial.print(I);

Serial.print(" D="); Serial.print(D);

Serial.print(" reading="); Serial.println(reading);

}

For **Pico/Python**:

#Configure the potentiometer pins for ADC input

potentiometerP=ADC(Pin(28))

potentiometerI=ADC(Pin(26))

potentiometerD=ADC(Pin(27))

#extra main control loop code:

#get and scale values from potentiometers

P=potentiometerP.read\_u16()/65536 #0-1

I=potentiometerI.read\_u16()/65536/100 #0-0.01

D=10\*potentiometerD.read\_u16()/65536 #0-10

print("P=%3.2f I=%3.4f D=%3.1f reading=%2d" % (P, I, D, reading))

#### Loop timing

Finally, in this section, add some timing control at the end of the control loop so the servo is only updated once every 20 milliseconds (50Hz).

For **Arduino/C++**:

//update servo at most once every 20ms

while(micros()-loopStart<20000);

For **Pico/Python**:

#update servo at most once every 20ms

while ticks\_us()-loopStart<20000:

pass

Note any difficulties you had in this section and describe how you overcame them.

If you did not experience any problems, explain how you know that your system is working correctly at this point.

|  |
| --- |
| Sample answer: check the println in the serial monitor.  Listen and watch for the servo moving. |

## Implement algorithms and design programming code to drive mechatronic devices

You will now test some simple on/off closed loop control code to verify that you are able to control the beam in the correct direction depending which side of centre the ball is on. You will need to know the following values for your system:

* the reading from the sensor when the ball is at the centre of the beam
* the control signal value required by the servo to keep the beam level
* approximate control signal values required by your servo to move the beam either side of level.

Adapt and temporarily add the following code to your main control loop for testing.

The numbers shown will need adjusting for your system.

Adjust values so that the beam on your physical system moves in the correct direction when the ball is away from the centre.

When the ball is in the centre, the beam should be level.

For **Arduino/C++**:

int readingCentre=800, servoCentre=90, controlSignal;

if(reading>readingCentre+100) controlSignal=servoCentre\*1.5;

else if(reading<readingCentre-100) controlSignal=servoCentre/1.5;

else controlSignal=servoCentre;

servo.write(int(controlSignal));

For **Pico/Python**:

readingCentre=800 #distance sensor value when ball centred

servoCentre=4000 #value required to balance the beam

if reading>readingCentre+100:

controlSignal=servoCentre\*1.5

elif reading<readingCentre-100:

controlSignal=servoCentre/1.5

else:

controlSignal=servoCentre

servo.duty\_u16(int(controlSignal))

Record the values for readingCentre and servoCentre for your system below, then delete the temporary test code from your project.

|  |  |
| --- | --- |
| readingCentre value for your system | servoCentre value for your system |
|  |  |

## Develop simulations and prototypes of a potential mechatronic system to test programming code

Create or adapt a simulation to match your physical circuit if you have not already done so.

All wiring connections should be the same on your physical circuit.

This will allow you to copy and paste code between the simulated and physical circuits easily.

Record the URL of your own simulation below.

|  |
| --- |
| Simulation URL |
|  |

## Design, develop and implement programming code for a closed loop control system

You will now code the PID control algorithm in your simulated circuit.

Add global variables with your recorded values for servoCentre and readingCentre.

Also add global variables to store integer values for setPoint, lastError and integral.

Then add the following code into your main control loop:

For **Arduino/C++**:

int error=reading-setPoint;

integral=integral+error;

int difference=error-lastError; //a real dE/dt derivative would divide the change in error by the change in time, but since the time difference is almost always exactly the same, we can save this part of the computation to speed up the calculation a little

float controlSignal=servoCentre+P\*error+I\*integral+D\*difference; //calculate control value

Serial.print(" P="); Serial.print(P);

Serial.print(" I="); Serial.print(I);

Serial.print(" D="); Serial.print(D);

Serial.print(" reading="); Serial.print(reading);

Serial.print(" error="); Serial.print(error);

Serial.print(" integral="); Serial.print(integral);

Serial.print(" difference="); Serial.print(difference);

Serial.print(" control signal="); Serial.println(controlSignal);

servo.write(int(controlSignal));

lastError=error;

For **Pico/Python**:

error=reading-setPoint

integral=integral+error

difference=error-lastError #a real dE/dt derivative would divide the change in error by the change in time, but since the time difference is almost always exactly the same, we can save this part of the computation to speed up the calculation a little

controlSignal=servoCentre+P\*error+I\*integral+D\*difference #calculate control value

print("P=%3.2f I=%3.4f D=%3.1f reading=%2d error=%2d integral=%2d difference=%2d control signal=%2d" % (P, I, D, reading, error, integral, difference, controlSignal))

servo.duty\_u16(int(controlSignal))

lastError=error

### Apply programming code to integrate sensors, actuators and end effectors/manipulators

Apply the code from your simulated circuit to your physical circuit.

After testing, you may notice that the system doesn’t behave as cleanly as you may have expected.

In the space below, suggest some possible reasons for any strange behaviours noted in your circuit.

|  |
| --- |
| Sample answer:  The servo makes the beam jump about in a jittery fashion.  This might be because the ultrasonic distance sensor is returning seemingly erratic values. Much of the power in the sound waves that bounce off the ball will be scattered at different angles rather than going directly back to the receiver.  When the ball finally stabilises near the centre, integral control takes some time to move the beam enough for the ball to actually move  By the time the ball starts moving again, the beam is often angled too high and so then overshoots the set point.  This could be because of frictional forces in the system.  The beam also doesn’t respond instantaneously when the ball is moving.  This could be caused by a range of factors including the response time of the ultrasonic sensor, the limited update speed of the servo, and physical slack in parts of the system (such as servo gearing and the servo-beam connection). |

### Implement specific control algorithms that enhance the performance of a mechatronic system

#### Setpoint changes

Allow the user to change the setpoint while the system is in operation by pressing the left, centre or right button.

This should see the system respond in an attempt to move the ball to the left, to the centre or to the right.

Insert code to initialise each button as a digital input.

Then adapt the pseudocode below and add to the main control loop or create your own code for your system.

IF left button is pressed THEN

setPoint=readingCentre\*2.0/3.0

ELSE IF centre button is pressed THEN

setpointPoint=readingCentre

ELSE IF right button is pressed THEN

setpoint=readingCentre\*3.0/2.0

END IF

#### Avoid integral windup

Integral windup is where the total integral value becomes so large that it takes a significant amount of time to overcome.

Integral windup is a common problem in PID control but there are a variety of techniques that can be used to limit its influence.

Adjust code in your system to limit integral windup. Some ideas you can try include:

* resetting the integral value to 0 whenever a button is pressed
* resetting the integral value to 0 whenever consecutive readings show that the ball has crossed the current setpoint value
* limit the use of integration only to values close to the current setpoint.

Describe the techniques you used to avoid integral windup and their effect on your system:

|  |
| --- |
| Sample answers could include:  I reset the integral value to 0 whenever a button is pressed which helped stabilise the system if it went into oscillations, for example:  #reset integrals if user changes the setPoint  if buttonLeft.value()==0:  setPoint=readingCentre\*2/3  integral=0  if buttonCentre.value()==0:  setPoint=readingCentre  integral=0  if buttonRight.value()==0:  setPoint=readingCentre\*3/2  integral=0  I also tried limiting integration to 20% from the readingCentre and increased the integral scaling factor but the readings were too erratic to tell if this was working well, for example:  if abs(error)>readingCentre/5:  integral=integral+error |

### Design, develop and implement a user interface (UI) to control a mechatronic system

#### Add a light-emitting diode (LED) for status information

It would be helpful for the user to know when the system thinks the ball is at, or very close to, the current setpoint.

An LED is appropriate for this type of status information that may change very quickly as it doesn’t take a substantial time to turn on or off.

You will need to connect a resistor and a LED in series from a digital output pin to ground. Watch this short video showing how to [calculate a LED resistor value](https://www.youtube.com/shorts/IDuseIqPOho).

After selecting and connecting a resistor and LED to your circuit, add code to output a digital HIGH when the error is small.

You should see the LED turn on whenever the ball is close to the current setpoint.

Record the code used to turn on your LED in the space below.

|  |
| --- |
| Sample answers could include:  For Arduino/C++:  if(abs(error)<setPoint/10) digitalWrite(LED,HIGH);  else digitalWrite(LED,LOW);  For Pico/Python:  if abs(error)<setPoint/10:  LED.on()  else:  LED.off() |

#### Add audible beeps for push-button feedback

Users would avoid repeated button pressing if the system gave them feedback to verify when it understood that a button had been pressed.

A buzzer is appropriate for this type of information as the delay used in making a sound can help prevent unintended consecutive button presses.

Connect a buzzer to your system and make it beep when any of the buttons are pressed.

Record the code used to create a beep in the space below.

|  |
| --- |
| Sample answers could include:  For Arduino/C++:  //reset integrals and beep if user changes the setPoint  if(digitalRead(left)==0){  setPoint=readingCentre\*2.0/3.0;  integral=0;  beep(1);  }  if(digitalRead(centre)==0){  setPoint=readingCentre;  integral=0;  beep(1);  }  if(digitalRead(right)==0){  setPoint=readingCentre\*3.0/2.0;  integral=0;  beep(1);  }  For Pico/Python:  #reset integrals and beep if user changes the setPoint  if buttonLeft.value()==0:  setPoint=readingCentre\*2/3  integral=0  beep(1)  if buttonCentre.value()==0:  setPoint=readingCentre  integral=0  beep(1)  if buttonRight.value()==0:  setPoint=readingCentre\*3/2  integral=0  beep(1) |

Suggest further user interface improvements for this system.

|  |
| --- |
| Sample answer:  Like the programmable articulated robot arm, the interface needs to be more physically robust and contain some basic control information to let the user know what each button and potentiometer is for.  The system should also show something to the user to let them know when it is powered on. |

Record your completed code below.

|  |
| --- |
| Sample answers could include:  For Arduino/C++ ([Wokwi simulation](https://wokwi.com/projects/383583582654267393))  //Balance beam - experiment and adjust to suit your circuit  #include <Servo.h>  Servo servo;  int trigger=2, echo=3;  int potentiometerP=A1, potentiometerI=A2, potentiometerD=A3;  int left=10, centre=11, right=12;  int readingCentre, servoCentre, setPoint, integral, lastError;  int LED=4;  void setup()  {  //Configure the potentiometer pins for ADC input  pinMode(potentiometerP,INPUT);  pinMode(potentiometerI,INPUT);  pinMode(potentiometerD,INPUT);    //configure pins for the ultrasonic distance sensor  pinMode(trigger,OUTPUT);  pinMode(echo,INPUT);    //the servo library sets the 50Hz pulse frequency  servo.attach(7, 1000, 2000);    //Configure pushbuttons for digital input with pullup resistors  pinMode(left,INPUT\_PULLUP);  pinMode(centre,INPUT\_PULLUP);  pinMode(right,INPUT\_PULLUP);    //set some initial variables - calibrate these for your balance beam and sensor  readingCentre=800; //the value given by the distance sensor when the ball is in the centre  servoCentre=90; //the control signal to send to the servo to keep the platform horizontal  setPoint=readingCentre; //the initial desired setpoint  integral=0;  lastError=0;  Serial.begin(9600);  }  int singleReading() {  //return a single unfiltered read from a sensor as a proxy for distance  digitalWrite(trigger, HIGH);  delayMicroseconds(5);  digitalWrite(trigger, LOW);  //measure the response time  int duration = pulseIn(echo, HIGH);  return duration;  }  void beep(int numBeeps){  //Turn the buzzer on and off a given number of times  for(int i=0; i<numBeeps; i=i+1){  tone(9,1000);  delay(300);  noTone(9);  delay(200);  }  }  void loop(){ //main control loop  int loopStart=micros();  //get values from potentiometers  float P=analogRead(potentiometerP)/1024.0/10.0; //0-0.1  float I=analogRead(potentiometerI)/1024.0/1000.0; //0-0.001  float D=analogRead(potentiometerD)/1024.0; //0-1  //reset integrals and beep if user changes the setPoint  if(digitalRead(left)==0){  setPoint=readingCentre\*2.0/3.0;  integral=0;  beep(1);  }  if(digitalRead(centre)==0){  setPoint=readingCentre;  integral=0;  beep(1);  }  if(digitalRead(right)==0){  setPoint=readingCentre\*3.0/2.0;  integral=0;  beep(1);  }  int reading=singleReading();  int error=reading-setPoint;  integral=integral+error;  int difference=error-lastError; //a real dE/dt derivative would divide the change in error by the change in time, but since the time difference is almost always exactly the same, we can save this part of the computation to speed up the calculation a little  float controlSignal=servoCentre+P\*error+I\*integral+D\*difference; //calculate control value  Serial.print(" P="); Serial.print(P);  Serial.print(" I="); Serial.print(I);  Serial.print(" D="); Serial.print(D);  Serial.print(" reading="); Serial.print(reading);  Serial.print(" error="); Serial.print(error);  Serial.print(" integral="); Serial.print(integral);  Serial.print(" difference="); Serial.print(difference);  Serial.print(" control signal="); Serial.println(controlSignal);  servo.write(int(controlSignal));  lastError=error;  if(abs(error)<setPoint/10) digitalWrite(LED,HIGH);  else digitalWrite(LED,LOW);  //update servo at most once every 20ms  while(micros()-loopStart<20000);  }  For Pico/Python ([Wokwi simulation](https://wokwi.com/projects/383583559803698177)):  #Balance beam - experiment and adjust to suit your circuit  #use microsecond time measurements  from machine import Pin,ADC,PWM, time\_pulse\_us  from time import sleep, sleep\_us, ticks\_us  #Configure a servo pin for PWM output  servo=PWM(Pin(6))  servo.freq(50)  #Configure the potentiometer pins for ADC input  potentiometerP=ADC(Pin(28))  potentiometerI=ADC(Pin(26))  potentiometerD=ADC(Pin(27))  #Configure the button pins for digital input with pull-up resistors  buttonLeft=Pin(12,Pin.IN,Pin.PULL\_UP)  buttonCentre=Pin(11,Pin.IN,Pin.PULL\_UP)  buttonRight=Pin(10,Pin.IN,Pin.PULL\_UP)  #Configure pins for the ultrasonic distance sensor  trigger=Pin(16,Pin.OUT)  echo=Pin(17,Pin.IN)  buzzer=PWM(Pin(14)) #Configure buzzer pin for PWM output  LED=Pin(1,Pin.OUT) #Configure an LED for digital output  def singleReading(timeOut=5000):  '''return a single unfiltered read from a sensor as a proxy for distance'''  startTime=ticks\_us()  trigger.on()  sleep\_us(5)  trigger.off()  while echo.value()==0 and ticks\_us()-startTime<timeOut:  pass  startTime=ticks\_us()  while echo.value()==1 and ticks\_us()-startTime<timeOut:  pass  if ticks\_us()-startTime>timeOut:  return -1 #bad read - took too long  else:  return min(ticks\_us()-startTime,2000)  def beep(numBeeps):  '''Make the buzzer beep a given number of times'''  buzzer.freq(1000) #Set the frequency  for i in range(numBeeps):  buzzer.duty\_u16(1000)  sleep(0.2)  buzzer.duty\_u16(0)  sleep(0.1)  #set some initial variables - calibrate these for your balance beam and sensor  readingCentre=800  setPoint=readingCentre  servoCentre=4000  integral=0  lastError=0  #main control loop  while True:  loopStart=ticks\_us()    #get values from potentiometers  P=potentiometerP.read\_u16()/65536 #0-1  I=potentiometerI.read\_u16()/65536/100 #0-0.01  D=10\*potentiometerD.read\_u16()/65536 #0-10    #reset integrals and beep if user changes the setPoint  if buttonLeft.value()==0:  setPoint=readingCentre\*2/3  integral=0  beep(1)  if buttonCentre.value()==0:  setPoint=readingCentre  integral=0  beep(1)  if buttonRight.value()==0:  setPoint=readingCentre\*3/2  integral=0  beep(1)    reading=singleReading()  error=reading-setPoint  integral=integral+error  difference=error-lastError #a real dE/dt derivative would divide the change in error by the change in time, but since the time difference is almost always exactly the same, we can save this part of the computation to speed up the calculation a little    controlSignal=servoCentre+P\*error+I\*integral+D\*difference #calculate control value  print("P=%3.2f I=%3.4f D=%3.1f reading=%2d error=%2d integral=%2d difference=%2d control signal=%2d" % (P, I, D, reading, error, integral, difference, controlSignal))  servo.duty\_u16(int(controlSignal))  lastError=error  if abs(error)<setPoint/10:  LED.on()  else:  LED.off()  #update servo at most once every 20ms  while ticks\_us()-loopStart<20000:  pass |

# Appendix

## Equipment selection information

### Microcontrollers vs microcontroller boards

While a microcontroller is technically a single integrated circuit, the term is often applied to microcontroller boards such as the Arduino Uno, Raspberry Pi Pico, and Micro:Bit.

Because microcontroller boards expose the microcontroller’s pins, they are very easy to use when prototyping and are recommended for most classroom activities where you would like to be able to reuse the device.

Microcontroller chips generally need to be soldered to a circuit board so are used when manufacturing a device or creating products that need to be smaller than a microcontroller board would normally allow.

Microcontroller boards are more expensive but can be reused more easily in many different projects.

The term ‘microcontroller board’ is used throughout this document when referring to a microcontroller chip that is soldered into a circuit board such as an Arduino Uno or Raspberry pi Pico.

### Microcontroller board considerations

Microcontroller boards have differing connectors, numbers of analog and digital IO pins, ADC resolution, microprocessor frequencies, pinouts, onboard sensors, logic levels, current sourcing ability, language support, program upload methods, costs and more.

Most boards support enough functionality for small project ideas, but some specific factors you may wish to consider include:

* Online community: Some boards have a large community of developers, so this can be a consideration if needing examples or advice.
* Logic level: The signal voltage level of a microcontroller board should match the voltage on any sensors and actuators it interacts with. Mismatching the logic level between the board and connected devices can stop the signals from being interpreted correctly and can damage some circuitry. Common logic levels on popular microcontroller boards are 5V and 3.3V. Selecting a 3.3V board will allow you to work with some newer sensors and actuators but may require external driver circuitry to interface with older or higher voltage devices.
* Some 5V boards will allow you to interface directly with some small actuators such as micro servos without requiring a separate power supply.
* Cost: Microcontroller board price seems to be more strongly related to the physical size of the board rather than its age (bigger boards cost more and few seem to devalue significantly over time).

Typical microcontroller board prices range from $10 to $50 each. In resourcing for the Programming mechatronics unit, the total cost of components external to the microcontroller board is likely to be higher than the board itself. You may also need to factor in the cost of cables or adapters needed to connect your microcontroller to a computer for programming.

### Comments on specific microcontroller boards

* Arduino Uno: This series of large boards has a very strong online community with many examples of student-level projects and code.

The pins are exposed through pre-soldered sockets.

The R3 model uses a USB-B connector for programming while the current R4 model uses a USB-C connector.

They both include an extra DC jack as an extra power connector.

The Uno uses 5V logic, can source up to 8mA per pin (R4) or 20mA per pin (R3) and includes 6 analog inputs.

* Arduino Nano: This series of boards is small and sometimes doesn’t come with the header pins pre-soldered.

The basic version is cheap but more advanced boards in this family come with features including more powerful microprocessors, more RAM, onboard sensors and/or networking. Cheaper knock-off versions are available but require installation of appropriate driver software on student computers. The Nano uses 3.3V logic (Every, BLE, ESP32, IoT, RP2040) or 5V logic (plain Nano), can source 7-40mA per pin depending on the model and includes 8 analog inputs.

* Raspberry Pi Pico: This is a newer microcontroller board and sometimes doesn’t come with the header pins pre-soldered.

It has a relatively powerful microprocessor and more RAM than most other boards.

It includes non-volatile storage so projects can read or write data. Extra ground pins make it easier to connect more devices directly to the board. The Pico uses 3.3V logic, can source up to 2-12mA per pin and includes 3 analog inputs.

* Raspberry Pi: This is a complete small board computer with its own operating system that can be programmed directly on the device with a connected keyboard and monitor.

It includes ports for USB devices, Ethernet and HDMI and always comes with the pins pre-soldered. It can run complete applications, including different language compilers for the installed Operating System (OS) so has extra flexibility over normal microcontroller boards.

Interacting with a Raspberry Pi from another device requires the setup and use of a protocol such as SSH with supporting software. The Raspberry Pi uses 3.3V logic, can source up to 16mA per pin and does not include any analog inputs.

* Micro:Bit: This is a small but very robust device that has an edge connector instead of pins. Connecting to external sensors and actuators can be done through alligator clips but for greater stability it would be better to use a breakout cable or board with a breadboard. It includes a small range of useful sensors and a 5x5 LED display.

The Micro:Bit uses 3.3V logic, can source up to 5mA per pin and includes 3 analog inputs.

* Other systems with sensors and actuators such as Lego and Vex could be used for this unit but align less strongly with some areas of the syllabus including wiring diagrams, sensors and actuators, mathematics, and power considerations.

They offer a more reliable path to project success by trading off the need to understand some of the mechanics and electronics behind their operation.

A more complete microcontroller board comparison is available on [Wikipedia.](https://en.wikipedia.org/wiki/Comparison_of_single-board_microcontrollers)

## Sensors, actuators and other components

### Sensors and inputs

The syllabus specifically mentions motion sensors and light sensors so these should be used in the classroom wherever possible.

Other input devices that are generally useful, very inexpensive, and easy to use include buttons and potentiometers.

There is a large range of other input devices and sensors available including joysticks, accelerometers, gyroscopes, and temperature and sound level sensors.

The logic level of sensors should match that of your microcontroller board, especially for boards with 3.3V logic levels.

### Actuators, end effectors and outputs

The syllabus specifically mentions hydraulic actuators and grippers.

Hydraulic actuators can be very expensive so you may need to consider how useful they will be for class activities.

There are many types of gripper and some can be purchased for under $100.

They can be purchased with or without the required servo, constructed or in parts, and individually or as part of a kit including something to attach it to.

When purchased in an unconstructed kit form, there are often many nuts and bolts to connect. Some grippers can also be made by hand, laser cut or 3D printed very cheaply.

Other output devices that are generally useful, very inexpensive, and easy to use include LEDs and piezoelectric buzzers.

Servo, stepper and DC motors are a good way to bring movement to a project but often require a separate power supply and motor driver or controller board to operate.

### Supporting components

You may like to consider some of the following types of components:

* USB cables and adaptors to connect between computers and controller boards for power and programming.
* Breadboards and jumper wires to connect devices and allow more than one connection to a single pin.

Breadboards and hats are especially useful for projects that require access to more power or ground pins than provided on the microcontroller board or to help contain a mess of cables.

You may need a range of jumper wire lengths and connections including plug-plug, plug-socket, and socket-socket.

* Motor drivers or controllers for projects that include motion.

Motors often consume more power than can be supplied from each pin on a microcontroller board.

This can be handled by passing the controlling signal through a transistor-based circuit or a motor driver or controller connected to a separate power supply.

* Batteries, battery holders and power supplies for projects that require movement or do a lot of electrical or physical work.
* Battery packs with an on/off switch can save batteries.

## Software considerations

**Teacher note**: consider sharing the information in this section with your school’s Technical Support Officer (TSO) or Information Technology (IT) support personnel

This unit was designed with cross-platform and UDM-packaged software in mind.

Depending on your choice of microcontroller board, software that each student would need installed may include:

* [Thonny](https://thonny.org/) – for programming in microPython with compatible microcontrollers
* [Arduino IDE](https://www.arduino.cc/en/software) – for programming in C++ with compatible microcontrollers.

Links to simulations are provided on the [Tinkercad](https://www.tinkercad.com/) and [Wokwi](https://wokwi.com/) websites.

The Tinkercad simulations show Arduino and C++ while the Wokwi simulations show Raspberry Pi Pico and microPython.

# References

This resource contains NSW Curriculum and syllabus content. The NSW Curriculum is developed by the NSW Education Standards Authority. This content is prepared by NESA for and on behalf of the Crown in right of the State of New South Wales. The material is protected by Crown copyright.

Please refer to the NESA Copyright Disclaimer for more information <https://educationstandards.nsw.edu.au/wps/portal/nesa/mini-footer/copyright>.

NESA holds the only official and up-to-date versions of the NSW Curriculum and syllabus documents. Please visit the NSW Education Standards Authority (NESA) website <https://educationstandards.nsw.edu.au/> and the NSW Curriculum website [https://curriculum.nsw.edu.au](https://curriculum.nsw.edu.au/).

[Software Engineering 11–12 Syllabus](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview) © NSW Education Standards Authority (NESA) for and on behalf of the Crown in right of the State of New South Wales, 2022.

adafruit (n.d.) [‘Continuous Servos – Collin’s Lab Notes’ [video]](https://youtube.com/shorts/BuLn8khjGV8?si=CUMjLuNx5PJHw7KF)*, adafruit*, YouTube, accessed 15 January 2024.

adafruit (n.d.) [‘Electret Mic Teardown – Collin’s Lab Notes’ [video]](https://youtube.com/shorts/IDuseIqPOho?si=cNam3yklhChKy8BF)*, adafruit*, YouTube, accessed 15 January 2024.

adafruit (n.d.) [‘Servo Motors – Collin’s Lab Notes’ [video]](https://youtube.com/shorts/TfakrB1IzbM?si=ExsXPLUhKCN3AH-A)*, adafruit*, YouTube, accessed 15 January 2024.

adafruit (n.d.) [‘Stacking Headers – Collin’s Lab Notes’ [video]](https://youtube.com/shorts/pR0w9-BBn1M?si=bDiB_8xVO3iMQOA4)*, adafruit*, YouTube, accessed 15 January 2024.

adafruit (n.d.) [‘Ultrasonic Distance Sensor – Collin’s Lab Notes’ [video]](https://youtube.com/shorts/O5l7IOmnD2Q?si=khFv4BqtD65nZkPb)*, adafruit*, YouTube, accessed 15 January 2024.

adafruit (n.d.) ['Phototransistor – Collin's Lab Notes' [video]](https://youtube.com/shorts/5HKvRrVWgYs?si=9h_7nXORYehect0A), *adafruit*, YouTube. accessed 15 January 2024.

Adafruit Industries (n.d.) ['Voltage Divider – Collin's Lab Notes' [video]](https://youtube.com/shorts/lQnDAs-igQM?si=WxgmGeS7dhw22BOA), *adafruit*, YouTube accessed 15 January 2024.

Adafruit Industries (20 May 2009) ['Using a piezo with an Arduino & photoresistor' [video]](https://youtu.be/Cok-FRz_-vc?si=4m2nodwklSmP4Aq7), *Adafruit Industries*, YouTube, accessed 15 January 2024.

Adafruit Industries (29 September 2021), ['Calculate LED Resistor Value - Collin’s Lab Notes' [video]](https://youtu.be/IDuseIqPOho), *Adafruit Industries,* YouTube, accessed 15 January 2024.

All About Circuits (n.d.) [‘Principles of digital computing: Microprocessors’](https://www.allaboutcircuits.com/textbook/digital/chpt-16/microprocessors/), All About Circuits website, accessed 15 January 2024.

API Metrology (29 June 2019) ['What Is 6 Degrees of Freedom?' [video]](https://youtu.be/DdvBrKl3SHg?si=ZdQg6RBhvEKa47Ih), *API Metrology*, YouTube accessed 15 January 2024.

Arduino (n.d.) [Downloads](https://www.arduino.cc/en/software), Arduino website, accessed 15 January 2024.

Arduino (n.d.) [Editor](https://create.arduino.cc/editor), Arduino website, accessed 15 January 2024.

Arduino (2024) [Analog Read Serial](https://docs.arduino.cc/built-in-examples/basics/AnalogReadSerial/), Arduino docswebsite, accessed 15 January 2024.

Arduino (2024) [Digital Input Pull-Up Resistor](https://docs.arduino.cc/tutorials/generic/digital-input-pullup/), Anduino docs website, accessed 15 January 2024.

Arduino (2024) [Inter-Integrated Circuit (I2C) Protocol](https://docs.arduino.cc/learn/communication/wire/), Arduino docs website, accessed 15 January 2024.

Arduino Projects & Robotics Tutorials (24 June 2022) ['Sensors Explained - What is a Sensor - Different Types | Robotics tutorial for Beginners' [video]](https://youtu.be/gizihSJ63o4?si=RiGPtGz4FxX58GYK), *Arduino Projects & Robotics Tutorials*, YouTube, accessed 15 January 2024.

Asim Zulfiqar (2022) [Button with Pi Pico](https://highvoltages.co/tutorial/pi-pico/button-with-pi-pico/,), High Voltages website, accessed 15 January 2024.

Autodesk (n.d.) [Circuits](https://www.tinkercad.com/circuits), Tinkercad website, accessed 15 January 2024.

Autodesk (n.d.) [Tinkercad](https://www.tinkercad.com/) [website], accessed 15 January 2024.

brmarcum (n.d.) [Microcontroller Register Manipulation](https://www.instructables.com/Microcontroller-Register-Manipulation/), Autodesk Instructables website, accessed 15 January 2024.

Business Insider (7 October 2015) ['Meet 'Iceman' and 'Wolverine' — the 2 coolest robots in Tesla's factory' [video]](https://youtu.be/WYnOGAvQEgk?si=P39HeBAkr4rWjUbi), *Business Insider*, accessed YouTube, 15 January 2024.

Campbell Scott (n.d.) [How to Control LEDs On the Arduino](https://www.circuitbasics.com/how-to-control-leds-on-the-arduino), Circuit Basics website, accessed 15 January 2024.

Chen Chien-Pin (9 February 2018) ['Demo of Redundant (7-dof) Robot Arm' [video]](https://youtu.be/BZnzIHdAoGo?si=Odjl2X7zrjF8h3UR), *Chien-Pin-Chen*, YouTube accessed 15 January 2024.

Chinmaya A.S.V (16 October 2018) ['Open Loop and Closed Loop Systems | Concept Video' [video]](https://youtu.be/WXDfOTlv4xE?si=xQYEM_EXunI3yrri), *Chinmaya A.S.V*, YouTube, accessed 15 January 2024.

Chrome for Developers (17 July 2017),['Gyroscope (The Standard, Ep. 10)' [video]](https://youtu.be/b_dTHSQVOZM?si=13a7HkJ0WEqAhYkO), *Chrome for Developers*, YouTube, accessed 15 January 2024.

CodeMagic LTD (n.d.) [Wokwi](https://wokwi.com/) [website], accessed 15 January 2024.

Core Electronics (n.d.) [9g Micro Servo - FS90 (180 degree, 1.5kg/cm)](https://core-electronics.com.au/feetech-fs90-1-5kgcm-micro-servo-9g.html), Core Electronics website, accessed 15 January 2024.

Core Electronics (n.d.) [Hobby Motor - Gear](https://core-electronics.com.au/hobby-motor-gear.html), Core Electronics website, accessed 15 January 2024.

Core Electronics (n.d.) [Photodiode](https://core-electronics.com.au/mini-photocell.html), Core Electronics website, accessed 15 January 2024.

Core Electronics (n.d.) [Robot Gripper - Extra Width](https://core-electronics.com.au/robot-gripper-extra-width.html), Core Electronics website, accessed 15 January 2024.

Core Electronics (n.d.) [Triple Axis Accelerometer ADXL345](https://core-electronics.com.au/triple-axis-accelerometer-adxl345.html), Core Electronics website, accessed 15 January 2024.

CSUMechatronics (13 May 2013) ['Coin Sorting Bank Mechatronics Project' [video]](https://youtu.be/K6YKykkaJEo?si=aCVh8_uA4cGBzB-C), *CSUMechatronics*, YouTube, accessed 15 January 2024.

CSUMechatronics (16 December 2015) ['Mechatronics Project - Self-Playing Guitar' [video]](https://youtu.be/3a5GoUoeDbY?si=qJHP52Z7R126RYyk), *CSUMechatronics*, YouTube, accessed 15 January 2024.

CSUMechatronics (17 May 2015) ['Automated Greenhouse' [video]](https://youtu.be/MziF7lOGO-U?si=IFDTylCphqGCBvK2), *CSUMechatronics,* YouTube, accessed 15 January 2024.

CSUMechatronics (2013) ['Mechatronics Project - "Micro Jackson" Dancing Robot - F13' [video]](https://youtu.be/r6Tb2ztNC40?si=dcWk89iaz0Iu09yZ), *CSUMechatronics,* YouTube, accessed 15 January 2024.

CSUMechatronics (9 May 2016) ['Nerf Ball Gun Turret - Mechatronics Project' [video]](https://youtu.be/S_RruxnvONQ?si=NticPwXEMfxqhjKe), *CSUMechatronics*, YouTube, accessed 15 January 2024.

CTaylor (n.d.) [Voltage Current Resistance And Ohms Law](https://learn.sparkfun.com/tutorials/voltage-current-resistance-and-ohms-law), sparkfun website, accessed 15 January 2024.

Dejan (n.d.) [How to Control Servo Motors with Arduino](https://howtomechatronics.com/how-it-works/how-servo-motors-work-how-to-control-servos-using-arduino/,), How To Mechatronics website, accessed 15 January 2024.

Dejan (n.d.) [Stepper Motors and Arduino – The Ultimate Guide](https://howtomechatronics.com/tutorials/arduino/stepper-motors-and-arduino-the-ultimate-guide), How To Mechatronics website, accessed 15 January 2024.

DigiKey (13 July 2022) ['How Hydraulic & Pneumatic Actuators Work – ATM | Digi-Key Electronics' [video]](https://youtu.be/95Y-WnbpQyk?si=n9-YnEQ4smZzbyB9), *DigiKey*, YouTube, accessed 15 January 2024.

Dorna Robotics (13 October 2023) ‘[Types of Robot Grippers and Their Applications](https://dorna.ai/blog/types-of-grippers-for-robots/,)’, *Dorna Robotics*, accessed 15 January 2024.

Douglas B (n.d.) ['What Is PID Control? | Understanding PID Control, Part 1' [video]](https://www.mathworks.com/videos/understanding-pid-control-part-1-what-is-pid-control--1527089264373.html), *Brian Douglas,* MathWorks website, accessed 15 January 2024.

DX mechanism (24 April 2020) ['10 hydraulic arm the principle of design mechanical mechanism' [video]](https://youtu.be/yI8E50Orkng?si=WIhCWevLfvd3t4Kv), *DX mechanism*, YouTube, accessed 15 January 2024.

Free the Data Academy and Sullins B (18 May 2023) ['Familiarity With Sensors And Actuators' [video]](https://www.linkedin.com/learning/top-10-skills-for-robotics-engineers/familiarity-with-sensors-and-actuators), *Top 10 Skills for Robotics Engineers*, LinkedIn Learning website, accessed 15 January 2024.

Free the Data Academy and Sullins B (18 May 2023) ['Skills for robotic engineers' [video]](https://www.linkedin.com/learning/top-10-skills-for-robotics-engineers/mechatronics,), *Top 10 Skills for Robotics Engineers*, LinkedIn Learning website accessed 15 January 2024.

Hardwarebee (n.d.) [Ultimate Guide: Microcontroller vs Microprocessor](https://hardwarebee.com/microcontroller-vs-microprocessor-what-to-choose/,), HardwareBee website, accessed 15 January 2024.

Hu Ryan (2 March 2018) ['I/O devices and peripherals' [video]](https://www.linkedin.com/learning/iot-foundations-operating-systems-fundamentals), *IoT Foundations: Operating Systems Fundamentals*, LinkedIn Learning website, accessed 15 January 2024.

Hydraulic and pneumatic systems (20 November 2021) ['How does a hydraulic cylinder work?' [video]](https://youtu.be/52IMMQSB9Hs?si=5VPgtrr856w7KsF3), *Hydraulic and pneumatic systems*, YouTube, accessed 15 January 2024.

Interface Chips (17 December 2014) ['Easy I2C: Introduction to I2C' [video]](https://youtu.be/qeJN_80CiMU?si=hUcvWFELTewSDlxN), *Interface Chips*, YouTube, accessed 15 January 2024.

Jimblom (n.d.) [Analog vs Digital,](https://learn.sparkfun.com/tutorials/analog-vs-digital?_gl=1*n49h8a*_ga*MTg2MDg5MDA4My4xNzA2NTA3ODUw*_ga_T369JS7J9N*MTcwNzk0NzI2OS4yLjEuMTcwNzk0NzI5OS4zMC4wLjA.) sparkfun website, accessed 15 January 2024.

Khalil Zara (28 February 2019) ['Understanding analog-to-digital converters' [video]](https://www.linkedin.com/learning/learning-arduino-interfacing-with-analog-devices), *Integrating analog devices with Arduino*, LinkedIn Learning website, accessed 15 January 2024.

Lemmon J (2023) [*Analog sensor data*](https://www.tinkercad.com/things/lw55awHDnqt?sharecode=-UZS-oQW6k1jH5IEBgzz9jMsTfcXYg9oeqs1D0r76ho)*,* Tinkercad website, accessed 15 January 2024.

Lemmon J (2023)[*Analog sensor data*](https://wokwi.com/projects/382497327363044353), wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Articulated arm (Arduino/C++): Complete*](https://wokwi.com/projects/383461457677997057), wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Articulated arm: button test*](https://www.tinkercad.com/things/iPnuvPMsZUx-articulated-arm-button-test?sharecode=cULREvCNCN7WSJv3fX8REIG5FIQuO859zCS8piIzvXE)*,* Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Articulated arm: button test*](https://wokwi.com/projects/383432370217383937), wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Articulated arm: Complete*](https://wokwi.com/projects/383437121741770753), wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Articulated arm: for student completion*](https://www.tinkercad.com/things/78I0AeUd9H2-articulated-arm-for-student-completion?sharecode=eM9nyRDK_4UWNGyryvvQoqT0rRl-LEqmSmibSd8yULQ), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Articulated arm: for student completion*](https://wokwi.com/projects/383435937768138753), wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Articulated arm: Potentiometer control*](https://www.tinkercad.com/things/4BidqOuVugn-articulated-arm-potentiometer-control?sharecode=ceO_nUTLKh-Q55Pn01c0frk0Y5AaSIQj8Hv84kTA8tY), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Articulated arm: potentiometer control*](https://wokwi.com/projects/383423687620008961), wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Articulated arm: servo control test*](https://wokwi.com/projects/383424942885241857), wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Articulated arm: Servo control*](https://www.tinkercad.com/things/hbJ1N0RiyBt-articulated-arm-servo-control?sharecode=Gv3a2W0XO6du-Gs29UeJUmamkQC1SnJQDvscaKrxfzo)*,* Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Balance beam - complete*](https://wokwi.com/projects/383583559803698177)*,* wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Balance beam (Arduino/C++) - complete*](https://wokwi.com/projects/383583582654267393)*,* wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Digital LED control*](https://www.tinkercad.com/things/bUvNVWQVqJB-digital-led-control?sharecode=0H2YU4cilmzjiKH_rk5TMv3OPQUXR6uaJ8HDqQkgk-s), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Digital LED control*](https://wokwi.com/projects/382497549868810241), wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Digital sensor data*](https://www.tinkercad.com/things/a3araTY082x-digital-sensor-data?sharecode=P4qKVqwGbk-sB3vowueUarlirk9eZE1VEU-eR9PaDXM), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Digital sensor data*](https://wokwi.com/projects/382497311139473409), wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Game: Automatic light avoidance*](https://www.tinkercad.com/things/hDRwBtaZDpT-game-automatic-light-avoidance?sharecode=FOR4RHhJPOjtU5doexn0EI_AtNutjGLlWiwFscTaUaw)*,* Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Game: Automatic light avoidance*](https://wokwi.com/projects/386516587027747841), wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Game: Complete*](https://www.tinkercad.com/things/aLDn5yK6n4O-game-complete?sharecode=zlaO-dWE3MdjyI9IQoszTdOtxgQxjUZilqE1d37eaXc), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Game: Complete*](https://wokwi.com/projects/382604400571376641)*,* wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Game: Test 2 servos*](https://www.tinkercad.com/things/bUYuyGgEijH-game-test-2-servos?sharecode=NsVVaIWtRDkhcSVZ_TzJueSxshMjzznhChvKA_M775o), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Game: Test 2 servos*](https://wokwi.com/projects/382593928813933569), wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Game: Test analog sensors*](https://www.tinkercad.com/things/duXYtpMv8qy-game-test-analog-sensors?sharecode=dYmC20cLwcMVFIoDqR75VNM44pZk5IZy5W8gL7e3MQg), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Game: Test analog sensors*](https://wokwi.com/projects/382596416920309761), wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Game: test buzzer*](https://www.tinkercad.com/things/1beKthYgagf-game-test-buzzer?sharecode=uuJayPBWwG4pJnziEL8Py7zLQtNFK3_fcZVXlnTh-DQ), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Game: test buzzer*](https://wokwi.com/projects/383247063303557121)*,* wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Game: Test LDR*](https://www.tinkercad.com/things/ePgpSwoZmBl-game-test-ldr?sharecode=JT0Wt29ntvtGpmO7FpT0u-tk-nJ8vIViwiFHwpH9Ww8), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Game: test LDR*](https://wokwi.com/projects/383244023092493313), wokwi website, accessed 15 January 2024.

Lemmon J (2023)[*Game: Test sensors and servos combined*](https://www.tinkercad.com/things/4NGqzYWvs3I-game-test-sensors-and-servos-combined?sharecode=ZpNWgnpts4i5eTDKpXANsIZsqWHrZm_YKcyW0JHkdXc), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Game: Test sensors and servos combined*](https://wokwi.com/projects/382599566948786177), wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*IMU data*](https://wokwi.com/projects/383981627952028673), wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*LCD text using I2C*](https://www.tinkercad.com/things/8agRp6pPU7B-lcd-text-using-i2c?sharecode=XNiWChCGgtRp2QxrVRa894hb85FPVF-9Ni88gOHlC2Q), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*LCD text using I2C*](https://wokwi.com/projects/382497225412085761), wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Light level plotting*](https://www.tinkercad.com/things/gWbFZoMwhnE?sharecode=kxnHu4hLMedNnYfom93991g_HgikKLTFraCgyh1CLfI), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Light level sensing*](https://www.tinkercad.com/things/4Yq9jDAaCkI?sharecode=w4PDMQTPzGxZ9cgso-bEShIZMJI0rK67yqlfIjWLlEQ), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Manual PWM rotational servo control*](https://www.tinkercad.com/things/86eSrMxNPDg?sharecode=7N6T6DVpwLiu9M3rlaX1xNK9kgoCo0h1hFVQcPRRGYM), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023)[*Manual PWM servo control*](https://wokwi.com/projects/382497269926238209)*,* wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Motion sensing with gyroscope*](https://wokwi.com/projects/383951631173430273), wokwi website, accessed 15 January 2024.

Lemmon J (2023) [*Motion sensing with PIR*](https://www.tinkercad.com/things/19GUtBCEtNs?sharecode=ruxIGep0oDyvWAWDytihpXMMmDPx--38WOmaUCtB_KU), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Motion sensing with ultrasonic distance sensor*](https://www.tinkercad.com/things/8WTJMFgO0IS-motion-sensing-with-ultrasonic-distance-sensor?sharecode=Nq_4BSLMgzDTaIGuaWmFWjOnBveaZ4pgZOTrcU8UEXc), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*PWM LED control*](https://www.tinkercad.com/things/lN4mABxxL6g?sharecode=TvqH-En4GxxRZDA36rRu1G52NzKjXg_R4EYyVSxXK7A)*,* Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*PWM motor control*](https://www.tinkercad.com/things/0PnkYS1yzkn?sharecode=kMRgNjICPwgxJ7w3l23uylng8jpdIxGLt_zm8p4adiA), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Servo pulse width control*](https://www.tinkercad.com/things/4zclfwr5M96-pwm-servo-pulse-width-control?sharecode=FgsEyH5eo53VFrgOGOM-9vjxBaNbDvpu_PtPM2NuVA4), Tinkercad website, accessed 15 January 2024.

Lemmon J (2023) [*Servo pulse width control*](https://wokwi.com/projects/382497346497462273), wokwi website, accessed 15 January 2024.

Lemmon J (2024) [Arduino Uno registers](https://wokwi.com/projects/387233104005573633), wokwi website, accessed 15 January 2024.

Lim H (5 February 2021) ['8 DoF Humanoid Robot Arm Control Test (Dynamixel Actuators)' [video]](https://youtu.be/iEeTZ0rD2yU?si=Iv-J1nOiAfq9fYTg), *Hwasup Lim,* YouTube, accessed 15 January 2024.

Make: (15 November 2011) ['Collin's Lab: Schematics' [video]](https://youtu.be/9cps7Q_IrX0?si=WgAullNFhlePxObT), *Make:*, YouTube, accessed 15 January 2024.

Makecode (n.d.) [Maker.Makecode.Com](https://maker.makecode.com)*,* Microsoft website, accessed 15 January 2024.

mcspanky893 (5 December 2015) ['Arduino PID vs Bang Bang Control of a Line Following Robot' [video]](https://youtu.be/t3R9cPq1aYk?si=tsf0wJBzEGKzbME6), *mcspanky893*, YouTube, accessed 15 January 2024.

MechWorks (2023) ['Bambu Labs P1S' [video]](https://youtube.com/shorts/3Pz3xrujm5w?si=LdRvLcBRH_DdaHbI), *MechWorks,* YouTube, accessed 15 January 2024.

MechWorks (2023) ['Final Warman Run' [video]](https://www.youtube.com/shorts/BKqxYLgQcDA), *MechWorks*, YouTube, accessed 15 January 2024.

Michigan Engineering (25 December 2015) ['Refreshable braille tablet that relies on microfluidics' [video]](https://www.youtube.com/watch?v=0fIg4rI4cDw), *Michigan Engineering,* YouTube, accessed 15 January 2024.

Microsoft MakeCode (n.d.) [Makecode.Microbit.Org](https://makecode.microbit.org) [website], accessed 15 January 2024.

Microsoft MakeCode (2 December 2018) ['Behind the MakeCode Hardware - Accelerometer on micro: bit' [video]](https://www.youtube.com/watch?v=byngcwjO51U), *Microsoft MakeCode*, YouTube, accessed 15 January 2024.

Microchip Technology Inc. (2020) [AVR Instruction Set Manual (PDF 1.14 MB)](https://ww1.microchip.com/downloads/en/DeviceDoc/AVR-Instruction-Set-Manual-DS40002198A.pdf), Microchip Technology Inc., accessed 15 January 2024.

mikeyearly (16 February 2018), ['Mini arduino simtools 2DOF motion sim platform' [video]](https://www.youtube.com/watch?v=ARU62KzuZM0), *mikeyearly*, YouTube, accessed 15 January 2024.

Nordic Semiconductor (n.d.) [*About Nordic Semiconductor*](https://www.nordicsemi.com/About-us) [website], accessed 15 January 2024.

NSW Education Standards Authority (NESA) (2023)  [Higher School Certificate Course Specifications Software Engineering (PDF 2.90 MB)](https://library.curriculum.nsw.edu.au/341419dc-8ec2-0289-7225-6db7f2d751ef/94e1eb0a-0df7-4dbe-9b72-5d5e0d17143a/software-engineering-11-12-higher-school-certificate-course-specifications.PDF)*, NESA,* accessed 15 January 2024.

Potent Printables (2018) [Linear Servo Actuators](https://www.thingiverse.com/thing:3170748), UltiMaker Thingiverse website, accessed 15 January 2024.

Potent Printables (28 October 2018) ['DIY Linear Servo Actuator, 3D Printed' [video]](https://www.youtube.com/watch?v=2vAoOYF3m8U), *Potent Printables*, YouTube, accessed 15 January 2024.

Progressive Automations (27 September 2013) ['Electric Linear Actuator - How It Works - Progressive Automations' [video]](https://www.youtube.com/watch?v=e9V2GY1WaRU), *Progressive Automations*, YouTube, accessed 15 January 2024.

Raspberry Pi Foundation (n.d.)[Introduction to Raspberry Pi Pico guide](https://projects.raspberrypi.org/en/projects/introduction-to-the-pico/11), Raspberry Pi Foundation website, accessed 15 January 2024.

Renishaw (14 April 2022) ['Six degrees of freedom (6DoF) explained' [video]](https://youtu.be/LHzkwt513q0), *Reinshaw*, YouTube, accessed 15 January 2024.

Rouse M (2019) [Six Degrees of Freedom](https://www.techopedia.com/definition/12702/six-degrees-of-freedom-6dof), Techopedia website, accessed 15 January 2024.

rubisco2000 (2018) [Snap-Together 9g Servo Gripper](https://www.thingiverse.com/thing:2775857), UltiMaker Thingiverse, accessed 15 January 2024.

Safarabadi M (26 July 2023) ['Linear Actuator' [video]](https://www.youtube.com/watch?v=stOcZqdEjmE), *Mohammad Safarabadi*, YouTube, accessed 15 January 2024.

SenstarCorp (13 December 2022) ['Motion Sensors' [video]](https://www.youtube.com/watch?v=SJDE36an3xk), *SenstarCorp*, YouTube, accessed 15 January 2024.

Şenvar Y (10 February 2022) ['BALANCING A BALL ON BEAM WITH PID USING ARDUINO AND PLOTTING GRAPH IN MATLAB' [video]](https://www.youtube.com/watch?v=tUDcpdTnhn4), *Şenvar Yigithan*, YouTube accessed 15 January 2024.

SFUptownMaker (n.d.) [I2C](https://learn.sparkfun.com/tutorials/i2c), sparkfun website, accessed 15 January 2024.

Simply Electronics (18 December 2016) ['What is an LDR? (Light Dependant Resistor) - Electronics Basics 26' [video]](https://www.youtube.com/watch?v=2fvXW4OEWLE), *Simply Electronics*, YouTube, accessed 15 January 2024.

skilledengineer (2023) ['How IR Detector Works?' [video]](https://www.youtube.com/shorts/3CvMyydiAlg), *skilledengineer,* YouTube, accessed 15 January 2024.

Thonny (n.d.) [Thonny.Org](https://thonny.org/) [website], accessed 15 January 2024.

TutorialsPoint (17 January 2018) ['Constrained Motion' [video]](https://youtu.be/OQbrwBbWfqo), *Tutorialspoint*, YouTube, accessed 15 January 2024.

Universal Robots (14 September 2020) ‘[Robot Grippers Explained](https://www.universal-robots.com/blog/robot-grippers-explained)’, *Universal Robots*, accessed 15 January 2024.

Valappil Ajith (14 July 2014) ['Android controlled 17 dof biped robot arduino + ssc-32 + JY-MCU' [video]](https://www.youtube.com/watch?v=n_1aVNnz9rA), *Valappil Ajith*, YouTube, accessed 15 January 2024.

Wikimedia Foundation, Inc. (2022) [Proportional Control](https://en.wikipedia.org/wiki/Proportional_control)*,* Wikipedia, accessed 15 January 2024.

Wikimedia Foundation, Inc. (2023) [Accelerometer](https://en.wikipedia.org/wiki/Accelerometer), Wikipedia, accessed 15 January 2024.

Wikimedia Foundation, Inc. (2023) [Atmega328](https://en.wikipedia.org/wiki/ATmega328)*,* Wikipedia, accessed 15 January 2024.

Wikimedia Foundation, Inc. (2023) [Memory Address Register](https://en.wikipedia.org/wiki/Memory_address_register), Wikipedia, accessed 15 January 2024.

Wikimedia Foundation, Inc. (2023) [Opcode](https://en.wikipedia.org/wiki/Opcode), Wikipedia, accessed 15 January 2024.

Wikimedia Foundation, Inc. (2023) [Processor Register](https://en.wikipedia.org/wiki/Processor_register)*,* Wikipedia, accessed 15 January 2024.

Wikimedia Foundation, Inc. (2023) [Proportional–integral–derivative controller](https://en.wikipedia.org/wiki/Proportional%E2%80%93integral%E2%80%93derivative_controller), Wikipedia, accessed 15 January 2024.

Wikimedia Foundation, Inc. (2023) [Robot end effector](https://en.wikipedia.org/wiki/Robot_end_effector)*,* Wikipedia, accessed 15 January 2024.

Wikimedia Foundation, Inc. (2023) [Rotary actuator](https://en.wikipedia.org/wiki/Rotary_actuator)*,* Wikipedia, accessed 15 January 2024.

Wikimedia Foundation, Inc. (2024) [Actuator](https://en.wikipedia.org/wiki/Actuator,)*,* Wikipedia, accessed 15 January 2024.

Wikimedia Foundation, Inc. (2024) [Bang–bang control](https://en.wikipedia.org/wiki/Bang%E2%80%93bang_control)*,* Wikipedia, accessed 15 January 2024.

Wikimedia Foundation, Inc. (2024) [Comparison of single-board microcontrollers](https://en.wikipedia.org/wiki/Comparison_of_single-board_microcontrollers)*,* Wikipedia, accessed 15 January 2024.

Wikimedia Foundation, Inc. (2024) [Control loop](https://en.wikipedia.org/wiki/Control_loop#Open-loop_and_closed-loop), Wikipedia, accessed 15 January 2024.

Wikimedia Foundation, Inc. (2024) [Linear actuator](https://en.wikipedia.org/wiki/Linear_actuator)*,* Wikipedia, accessed 15 January 2024.

Wikimedia Foundation, Inc. (2024) [Rp2040](https://en.wikipedia.org/wiki/RP2040)*,* Wikipedia, accessed 15 January 2024.

Wikimedia Foundation, Inc. (2024)[Sensor](https://en.wikipedia.org/wiki/Sensor)*,* Wikipedia, accessed 15 January 2024.

Woodford C (19 November 2021) ‘[Hydraulics](https://www.explainthatstuff.com/hydraulics.html)’, *explainthatstuff,* accessed15 January 2024.

아름다울 청년 (12 February 2019) ['Hydraulic Robot Arm' [video]](https://www.youtube.com/watch?v=3MR_O3xDAVU), 아름다울 청년, YouTube, accessed 15 January 2024.

**© State of New South Wales (Department of Education), 2024**

The copyright material published in this resource is subject to the Copyright Act 1968 (Cth) and is owned by the NSW Department of Education or, where indicated, by a party other than the NSW Department of Education (third-party material).

Copyright material available in this resource and owned by the NSW Department of Education is licensed under a [Creative Commons Attribution 4.0 International (CC BY 4.0) license](https://creativecommons.org/licenses/by/4.0/).

[](https://creativecommons.org/licenses/by/4.0/)

This license allows you to share and adapt the material for any purpose, even commercially.

Attribution should be given to © State of New South Wales (Department of Education), 2024.

Material in this resource not available under a Creative Commons license:

* the NSW Department of Education logo, other logos and trademark-protected material
* material owned by a third party that has been reproduced with permission. You will need to obtain permission from the third party to reuse its material.

**Links to third-party material and websites**

Please note that the provided (reading/viewing material/list/links/texts) are a suggestion only and implies no endorsement, by the New South Wales Department of Education, of any author, publisher, or book title. School principals and teachers are best placed to assess the suitability of resources that would complement the curriculum and reflect the needs and interests of their students.

If you use the links provided in this document to access a third-party's website, you acknowledge that the terms of use, including licence terms set out on the third-party's website apply to the use which may be made of the materials on that third-party website or where permitted by the Copyright Act 1968 (Cth). The department accepts no responsibility for content on third-party websites.