Mission 5: Hovering Flight

Time Frame: 90-180 minutes

Mission Goal: Students will program CodeAIR to avoid walls and seek an exit by utilizing its sensors.

Learning Targets

- I can use the console output print() statement.
- I can control CodeAIR with the MotionCommander interface.
- I can use blocking and non-blocking functions.
- I can measure distances with CodeAIR's laser rangers.
- I can work with variables in Python.

Key Concepts

- You can save important functions as a custom module. Then import and use the module in future programs.
- CodeAIR uses a high-level flight control interface called MotionCommander that uses sensors to maintain stable flight.
- Functions can be blocking or non-blocking.
- When it is connected to CodeSpace, CodeAIR can print information to the console.
- A variable is a name you attach to an object so your code can work with it.
- You can use a variable to give your code memory. Update the variable to use it as a counter.

Assessment Opportunities

- Quiz after Objective 5
- Quiz after Objective 7
- Quiz after Objective 9
- Save *safety.py* as a custom module
- Complete the program *Hover*
- Complete the program *Rangers*
- Complete the program *Ceiling*
- Complete the program *Theremin*
- Complete the program *HallMonitor*
- Complete the program *Avoidance*
- Mission 5 Assignment
- Mission 5 Review questions

Success Criteria

- ☐ Create the *safety.py* custom module
- ☐ Use MotionCommander commands to fly
- ☐ Read sensors and unpack their values into variables
- ☐ Use the CodeAIR and variables as a people counter
- ☐ Use the blue LEDs as counters
- ☐ *Avoidance* works correctly and runs without errors or bugs
- ☐ Complete Mission 5 Assignment

Standards

CSTA Standards Grades 9-10	CSTA Standards Grades 11-12	AI4K12 Standards Grades 9-12
• 3A-CS-03 • 3A-DA-11 • 3A-AP-13 • 3A-AP-15 • 3A-AP-16 • 3A-AP-17 • 3A-AP-18 • 3A-AP-21 • 3A-IC-24 • 3A-IC-26	• 3B-DA-06 • 3B-AP-10 • 3B-AP-14 • 3B-AP-15 • 3B-AP-16 • 3B-AP-21 • 3B-AP-22 • 3B-AP-23	• 1-A-ii

Student Materials

- Laptop/computer with Chrome browser
- CodeAIR drone and USB cable
- Poster board or similar material to use as a "wall"
- CodeAIR Mission 5 Assignment
- CodeAIR Flying Guide

Teacher Resources

- CodeAIR Mission 5 Assignment Answers
- CodeAIR Mission 5 Review Questions
- CodeAIR Flying Guide

Vocabulary

Module	An external source of code that is outside your own source file; also known as a library.
Custom module	Some code that is in the same folder as your program and can be accessed by importing it.
docstring	A documentation string; a comment at the top of the file that explains what it does. Use triple quotes (‘ ‘ ‘ or “ “ “) to start and stop a docstring.
Console	A window that lets you see output from print() statements.
Blocking function	A function that runs one line at a time, blocking your code from continuing until they are finished. Examples: steady() and sleep()
Non-Blocking function	A function that starts a movement, then returns to the code. Another command must be sent to change or stop the movement.
OODA loop	“Observe, orient, decide, act” – a continuous loop run by the CPU to keep the drone flying at a desired altitude.
Variable	A name attached to an object so your code can work with it. The object can be any data: a number, text, tuple, etc.
Tuple	Ranger data – a set of three values indicated with parenthesis (forward, up, down).
Polling	Repeatedly checking something to see if anything has changed.
Actuator	A device that receives signals and responds with a specific action. When flying a drone, the motors are actuators that receive input from sensors.
Updating a variable	Changing the value of a variable with assignment. An example is to increment a count by 1.
Dead reckoning	A type of navigation that calculates the vehicle’s position based on its known starting point, speed, direction and elapsed time.
Sensor-based navigation	A type of navigation that is adaptive, relying on real-time data gathered by sensors to detect obstacles and adjust course accordingly.
REPL	Repeat evaluate print loop; using the console to interactively enter commands and view outputs in a text format.

New Python Code

`'''This is a docstring'''`	Document string that should go at the top of any module
`fly.take_off(height_meters)`	Ascend to given height altitude
`fly.steady(seconds)`	Hover, allows code to pause while keeping the flight controller running
`fly.land()`	Descend to the floor
`fly.forward(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)
`fly.back(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)

`fly.left(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)
`fly.right(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)
`fly.up(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)
`fly.down(distance, velocity)`	Distance in meters, velocity in meters per second (defaults to 0.2)
`get_data(RANGERS)`	Returns the (forward, up, down) distance in millimeters
`if up < too_close:` `  # sound alarm`	If statement with a condition
`fwd, up, down = get_data(RANGERS)`	Unpack the data from the rangers from the three values in the tuple to three variables

`ticks = timeout * 10`
`for i in range(ticks):`
`  fly.steady(0.1)`
`  fwd, up, down = get_data(RANGERS)`
`  if up < too_close:`
`  return True`
`return False`
	Algorithm for polling with a blocking function.
	In this example, timeout is a parameter that receives seconds from an argument. The polling will happen ten times per second.

`speaker.beep(400, 0)`	Causes the beep to play continuously. Requires speaker.off() to stop the beep.
`count = count + 1`	Incrementing or updating a variable
`leds.set_mask(0, 0)`	Turn off all the blue LEDs
`fly.start_forward()`	Non-blocking function that starts moving forward at the default velocity and returns immediately so the next instruction can be executed
`fly.stop()`	Stop any motion and hover
`fly.turn_left(degrees)`	A blocking function that turns the drone degrees left
`if count == 8:`	Checks if **count** is the same as 8. If it is, a branch of code is executed.

Teacher Notes

- This mission is lengthy and includes several programs and concepts. Take your time, even an entire week. Using several days for the Objectives will give your students plenty of time to explore and try things.
- In Objective #1, students add code to their custom module. The *__name__* and '*__main__*' use two underscores _ _. You may need to mention this to students to avoid errors.
- The floor space and lighting really do make a difference in how well CodeAIR can track its movements. If you have a lot of drift, move to different flooring or add a pattern to the floor to help with navigation.
- In Objective #6, think safety first! Use a file folder or clipboard as an obstacle above the drone instead of a hand. You don't want fingers in the blades.
- The CodeTrek in Objective #8 shows a step-by-step algorithm for counting people. This mission uses several algorithms. This is a good time to have a discussion about algorithms and practice writing them.
- Extensions and cross-curricular projects are included to enhance the concepts in the mission. You can use the extensions to extend students' programming experience. A remix is planned after Mission 5.

Extensions	Cross-Curricular
• Create a more elaborate security system after Objective 5 and/or Objective 6. • Code a different ending to fix the bug after Objective 10. • Have students do a code review. Pick any of the six programs to review. • The programs in this mission use algorithms to solve a problem. Discuss what an algorithm is. Practice writing algorithms.	• **MATH:** In Objective #3 the drone flies forward a given distance. Create a chart of distances entered in code and actual distance flown. Calculate the difference. Write an equation to predict future flights. • **SCIENCE:** The theremin was invented during research into proximity sensors. Study the physics involved in the musical instrument. Try building and playing your own theremin. • **LANGUAGE ARTS:** Write a paragraph that explains when to use a loop and when to use an if statement in code. • **LANGUAGE ARTS**: Evaluate the ways computing impacts personal or cultural practices. • **LANGUAGE ARTS:** Make a list of troubleshooting strategies used to identify and fix errors.