Q1 SIMULATION ENVIRONMENT SETUP

Q2 SPIRAL PATH IMPLEMENTATION

#include <iostream>

#include <cmath>

#include <mavsdk/mavsdk.h>

#include <mavsdk/plugins/action/action.h>

#include<mavsdk/plugins/telemetry/telemetry.h>

Using namespace mavsdk;

Int main() {

Mavsdk mavsdk;

ConnectionResult connection\_result = mavsdk.add\_udp\_connection();

If (connection\_result != ConnectionResult::Success) {

Std::cout << “Failed to connect to the drone: “ << connection\_result << std::endl;

Return 1;

}

Auto system = mavsdk.systems().at(0);

Auto action = Action{system};

Auto telemetry = Telemetry{system};

Const double r = 10.0; // Radius of the spiral.

Const double n = 2.0; // Number of rotations per second.

Const double v\_f = 5.0; // Forward velocity of the spiral.

// Arm the drone

Const ActionResult arm\_result = action.arm();

If (arm\_result != ActionResult::Success) {

Std::cout << “Arming failed: “ << arm\_result << std::endl;

Return 1;

}

// Take off to a specified altitude

j Const ActionResult takeoff\_result = action.takeoff();

If (takeoff\_result != ActionResult::Success) {

Std::cout << “Takeoff failed: “ << takeoff\_result << std::endl;

Return 1;

}

// Command the drone to perform the spiral trajectory

Const double duration = 10.0;

Const auto start\_time = std::chrono::steady\_clock::now();

While (std::chrono::steady\_clock::now() – start\_time < std::chrono::seconds(static\_cast<int>(duration))) {

Const auto current\_time = std::chrono::steady\_clock::now() – start\_time;

Const double t = std::chrono::duration\_cast<std::chrono::milliseconds>(current\_time).count() / 1000.0;

Const double x = r \* std::sin(2 \* M\_PI \* n \* t);

Const double y = v\_f \* t;

Const double z = r \* std::cos(2 \* M\_PI \* n \* t);

// Command the drone to the desired position

Const ActionResult go\_to\_result = action.goto\_location(x, y, z, 0.0);

If (go\_to\_result != ActionResult::Success) {

Std::cout << “Failed to go to location: “ << go\_to\_result << std::endl;

Return 1;

}

Std::this\_thread::sleep\_for(std::chrono::milliseconds(100));

}

// Land the drone

Const ActionResult land\_result = action.land();

If (land\_result != ActionResult::Success) {

Std::cout << “Landing failed: “ << land\_result << std::endl;

Return 1;

}

// Disarm the drone

Const ActionResult disarm\_result = action.disarm();

If (disarm\_result != ActionResult::Success) {

Std::cout << “Disarming failed: “ << disarm\_result << std::endl;

Return 1;

}

Return 0;

}

Q3 ARUCO LANDING

#include <opencv2/aruco.hpp>

#include <opencv2/core.hpp>

#include <opencv2/highgui.hpp>

#include <iostream>

Using namespace cv;

Using namespace std;

Int main() {

// Initialize the video capture device

VideoCapture cap(0); // Use camera index 0 (default camera)

// Check if the video capture device is opened successfully

If (!cap.isOpened()) {

Cout << “Failed to open the camera.” << endl;

Return 1;

}

// Create a window to display the camera feed

namedWindow(“Camera Feed”, WINDOW\_AUTOSIZE);

// Initialize the ArUco detector

Ptr<aruco::Dictionary> aruco\_dict = aruco::getPredefinedDictionary(aruco::DICT\_4X4\_50);

Aruco::DetectorParameters parameters;

// Main loop

While (true) {

// Read a frame from the camera

Mat frame;

Cap.read(frame);

// Detect the ArUco markers in the frame

Vector<Vec3d> corners, rvecs, tvecs;

Vector<int> ids;

Aruco::detectMarkers(frame, aruco\_dict, corners, ids, parameters);

// If any ArUco markers were detected, get their positions and orientations

If (ids.size() > 0) {

// Estimate the pose of the ArUco markers

Aruco::estimatePoseSingleMarkers(corners, 0.05, cameraMatrix, distCoeffs, rvecs, tvecs);

// Draw the marker axes on the frame

For (size\_t I = 0; I < ids.size(); i++) {

Aruco::drawAxis(frame, cameraMatrix, distCoeffs, rvecs[i], tvecs[i], 0.1);

}

}

// Show the frame in the window

Imshow(“Camera Feed”, frame);

// Check for key press to exit the loop If (waitKey(1) == ‘q’) {

Break;

}

}

// Release the video capture device and close the window

Cap.release();

destroyAllWindows();

return 0;

}