



PerfectForm

Course Project

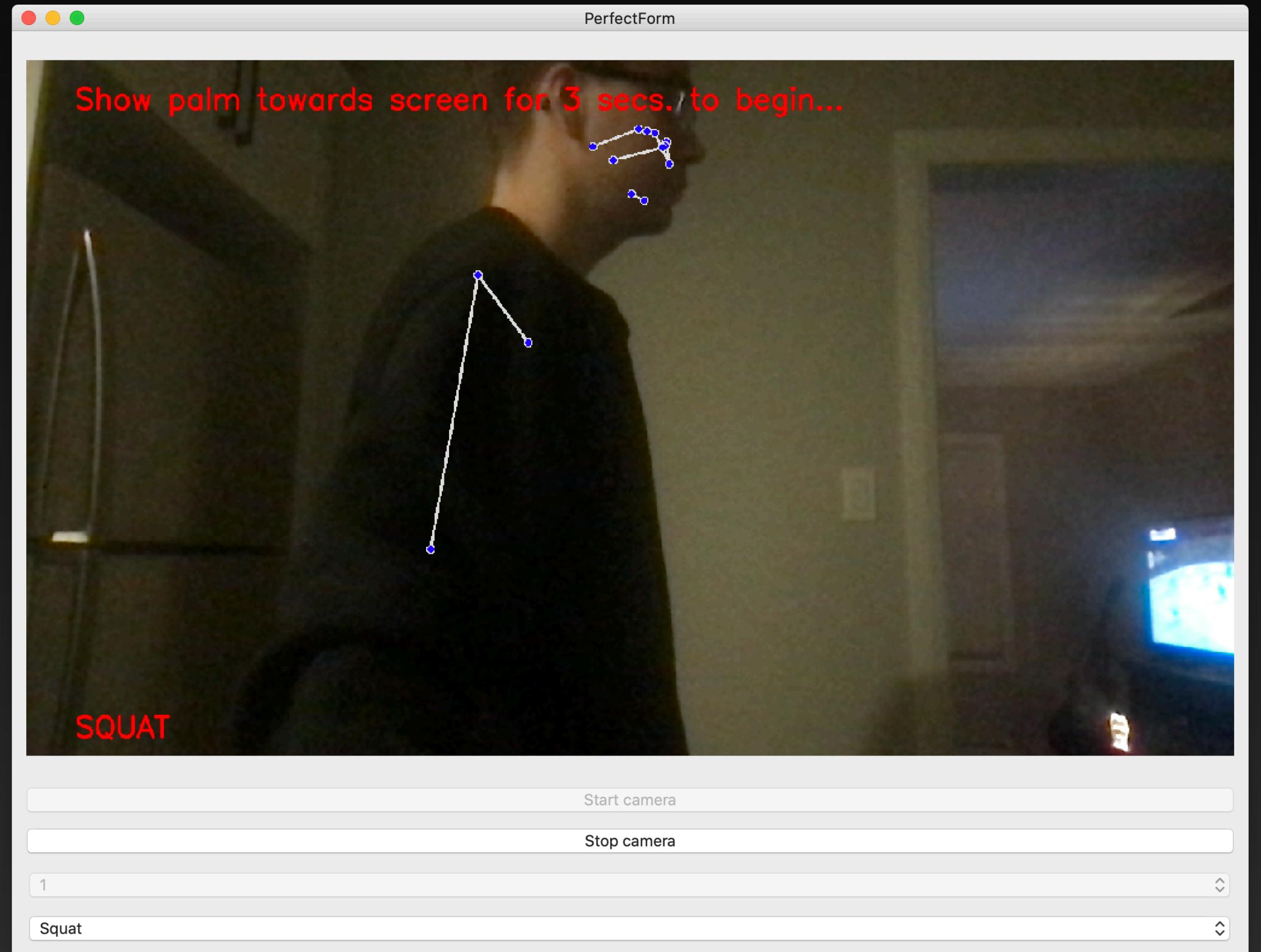
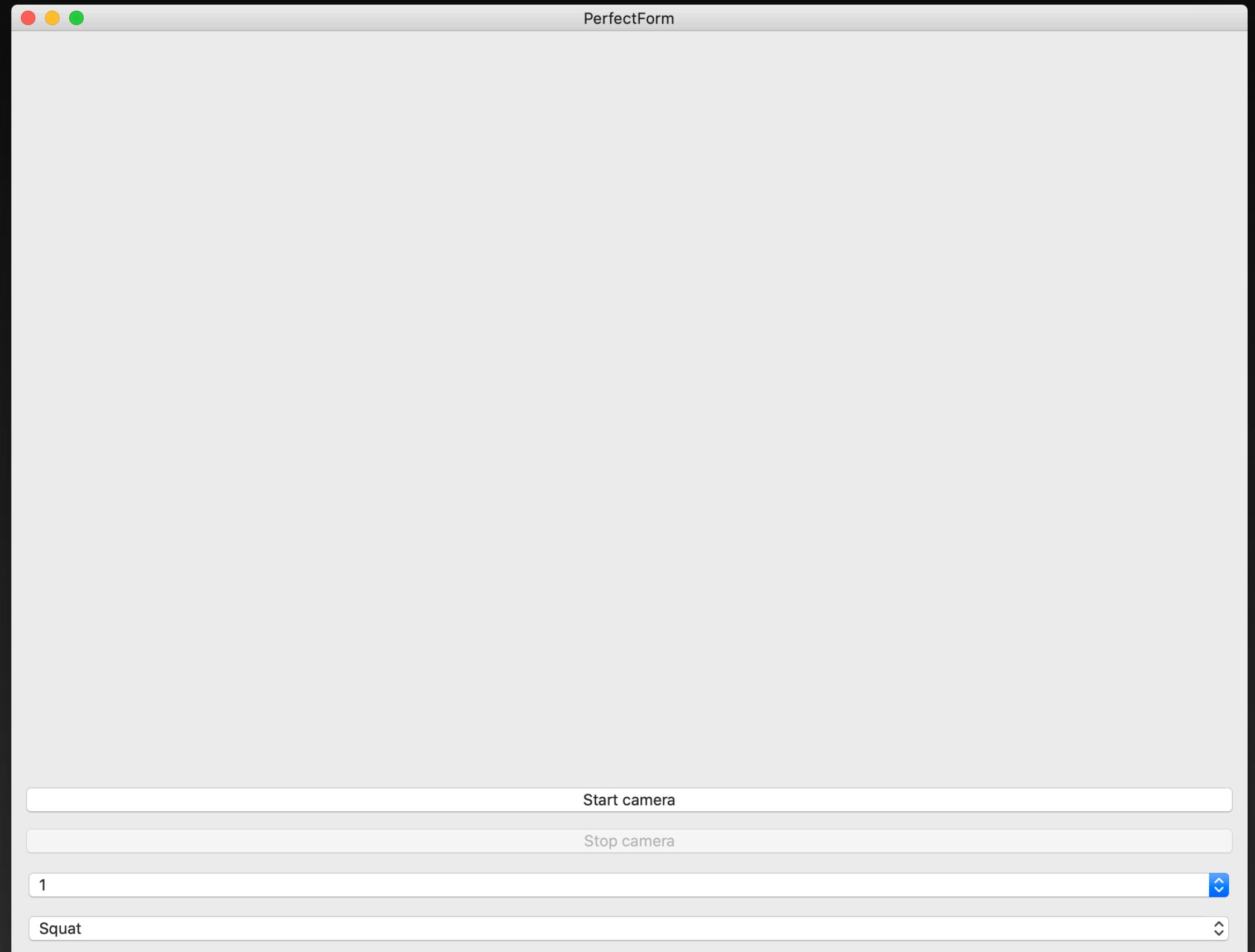
Computer Vision 4301

Divya Murali Krishnan, Tarun Subramanian, Jacob Critch

Overview

- Computer vision application for detecting form/posture in the context of physical exercises
- Primarily developed for squatting
- Built using OpenCV and MediaPipe Pose
- Python 3 application with PyQt5 GUI

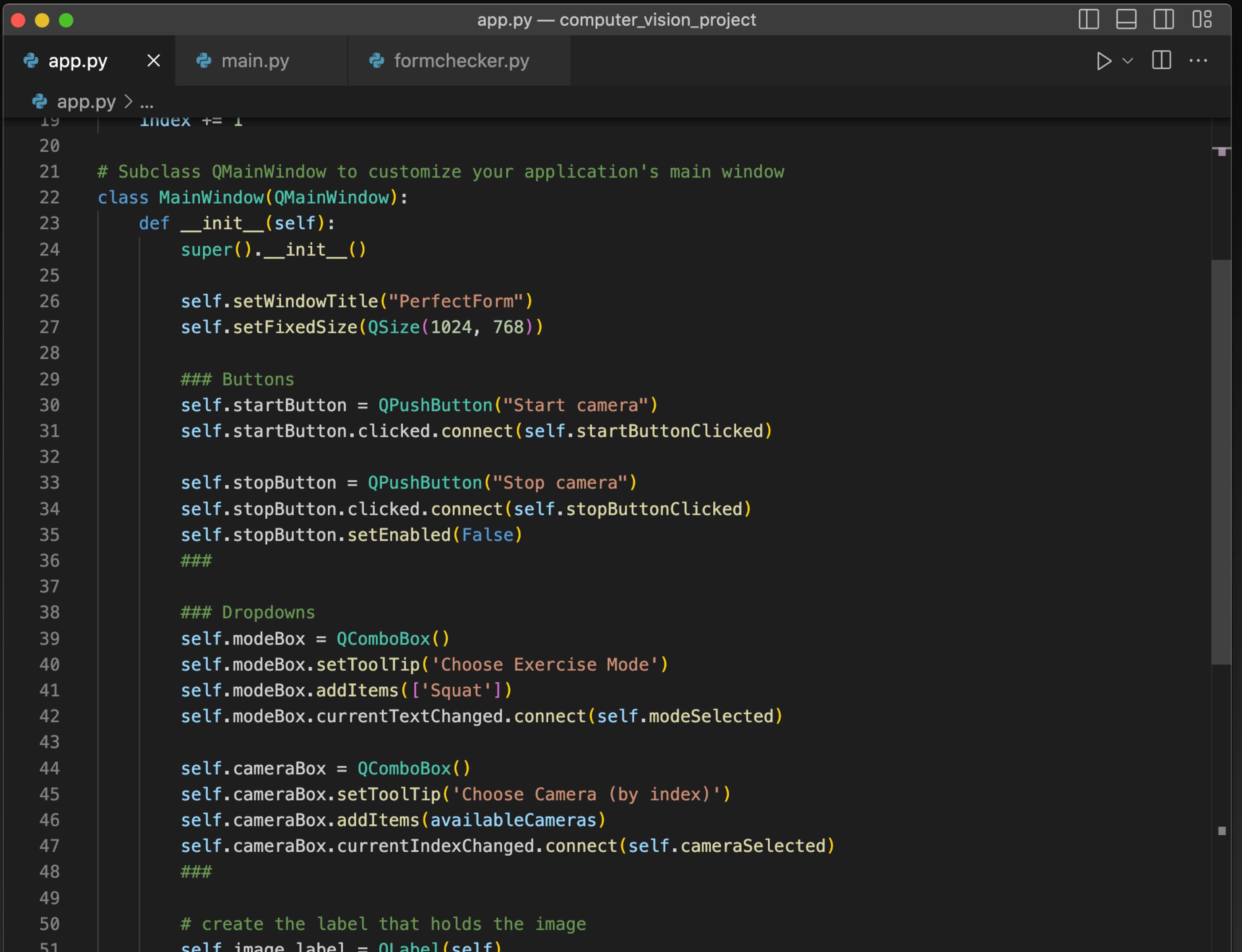
Overview



Methods and Tooling

- **MediaPipe** offers cross-platform, customizable ML solutions for live and streaming media
- The **MediaPipe Pose** Landmarker task lets you detect the landmarks of human bodies in an image. You can use this task to identify key body locations and render visual effects on them. This task uses machine learning (ML) models that can work with single images or a continuous stream of images.
- **OpenCV** provides a real-time optimized Computer Vision library, tools, and hardware.

Methods and Tooling App



The screenshot shows a code editor window titled "app.py — computer_vision_project". The tab bar includes "app.py", "main.py", and "formchecker.py", with "app.py" being the active tab. The code itself is a Python script using PyQt5 for the GUI. It defines a subclass of QMainWindow named MainWindow. The script handles button creation for starting and stopping the camera, and dropdown menus for selecting exercise modes and cameras. It also manages the current index of the selected camera.

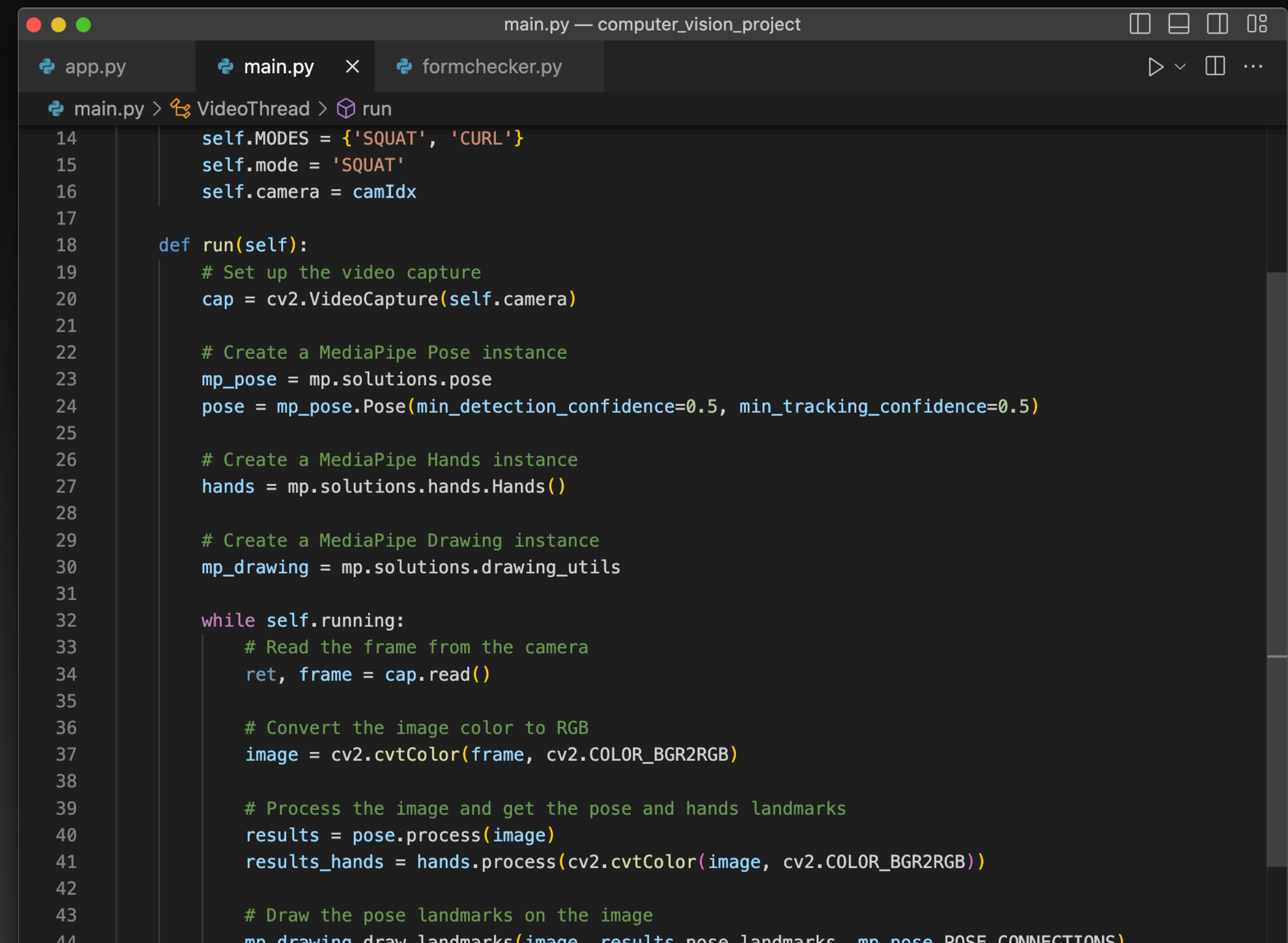
```
app.py — computer_vision_project
app.py      X  main.py  formchecker.py
app.py > ...
19     index += 1
20
21 # Subclass QMainWindow to customize your application's main window
22 class MainWindow(QMainWindow):
23     def __init__(self):
24         super().__init__()
25
26         self.setWindowTitle("PerfectForm")
27         self.setFixedSize(QSize(1024, 768))
28
29     ### Buttons
30     self.startButton = QPushButton("Start camera")
31     self.startButton.clicked.connect(self.startButtonClicked)
32
33     self.stopButton = QPushButton("Stop camera")
34     self.stopButton.clicked.connect(self.stopButtonClicked)
35     self.stopButton.setEnabled(False)
36     ###
37
38     ### Dropdowns
39     self.modeBox = QComboBox()
40     self.modeBox.setToolTip('Choose Exercise Mode')
41     self.modeBox.addItems(['Squat'])
42     self.modeBox.currentTextChanged.connect(self.modeSelected)
43
44     self.cameraBox = QComboBox()
45     self.cameraBox.setToolTip('Choose Camera (by index)')
46     self.cameraBox.addItems(availableCameras)
47     self.cameraBox.currentIndexChanged.connect(self.cameraSelected)
48     ###
49
50     # create the label that holds the image
51     self.image_label = QLabel(self)
```

PyQt5 GUI definition,
main thread controls secondary
OpenCV/MediaPipe thread,
camera selection

Methods and Tooling

Main

OpenCV methods to capture video feed, and apply processing (overlay, text) based on formchecker results

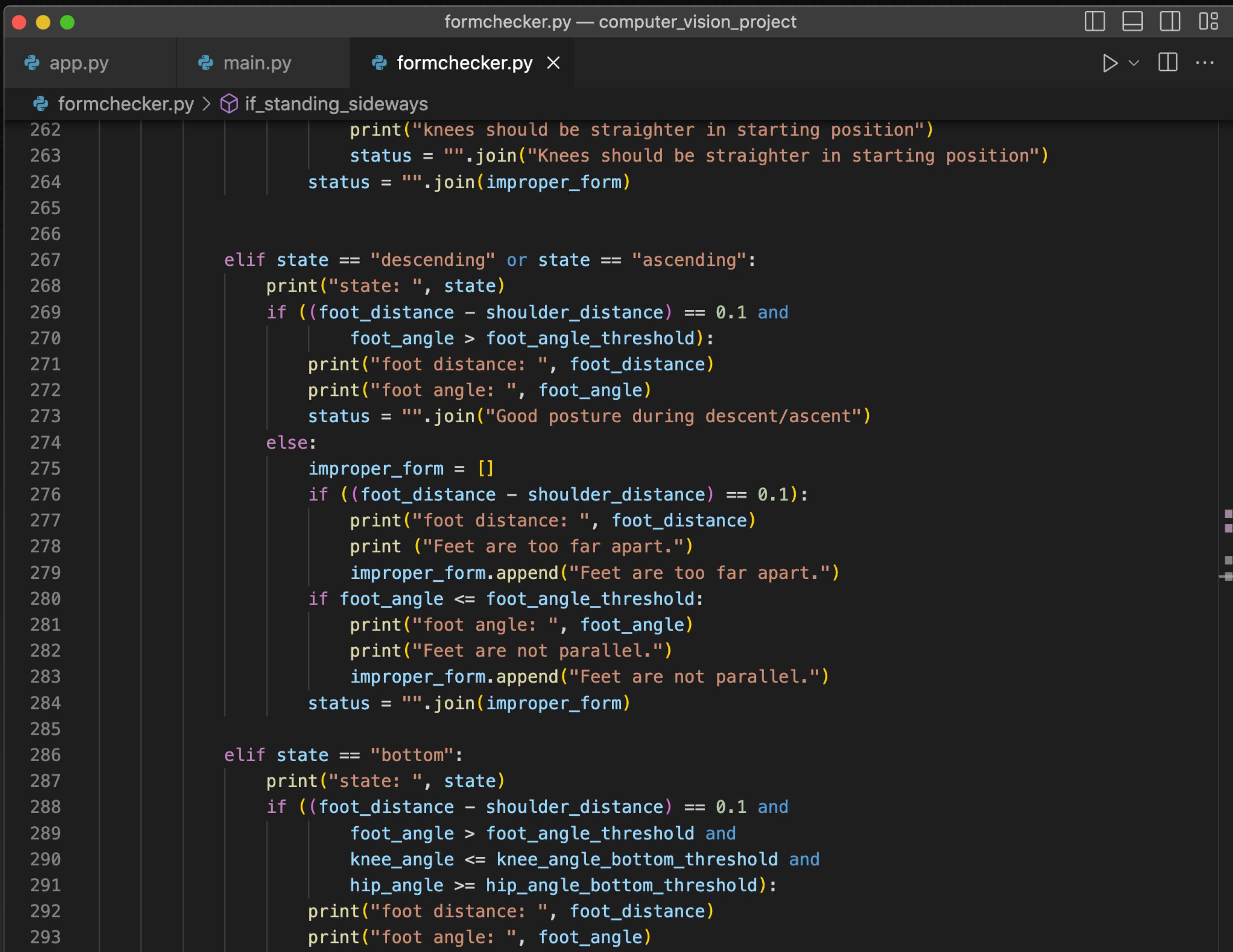


The screenshot shows a code editor window titled "main.py — computer_vision_project". The main.py file is open, showing a script that uses OpenCV and MediaPipe libraries to capture a video feed from a camera, process it for pose and hand landmarks, and draw those landmarks on the frame. The code includes imports for cv2, mp.solutions.pose, mp.solutions.hands, and mp.solutions.drawing_utils, along with definitions for MODES, camera, and a run() function that sets up the video capture, creates MediaPipe instances for pose and hands, and then enters a loop to read frames, convert them to RGB, process them for landmarks, and draw those landmarks on the image.

```
main.py — computer_vision_project
app.py    main.py    formchecker.py
main.py > VideoThread > run
14     self.MODES = {'SQUAT', 'CURL'}
15     self.mode = 'SQUAT'
16     self.camera = camIdx
17
18     def run(self):
19         # Set up the video capture
20         cap = cv2.VideoCapture(self.camera)
21
22         # Create a MediaPipe Pose instance
23         mp_pose = mp.solutions.pose
24         pose = mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5)
25
26         # Create a MediaPipe Hands instance
27         hands = mp.solutions.hands.Hands()
28
29         # Create a MediaPipe Drawing instance
30         mp_drawing = mp.solutions.drawing_utils
31
32         while self.running:
33             # Read the frame from the camera
34             ret, frame = cap.read()
35
36             # Convert the image color to RGB
37             image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
38
39             # Process the image and get the pose and hands landmarks
40             results = pose.process(image)
41             results_hands = hands.process(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
42
43             # Draw the pose landmarks on the image
44             mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS)
```

Methods and Tooling

Formchecker



The screenshot shows a dark-themed code editor window titled "formchecker.py — computer_vision_project". The window contains three tabs: "app.py", "main.py", and "formchecker.py X". The "formchecker.py" tab is active, displaying the following Python code:

```
formchecker.py > if_standing_sideways
262     print("Knees should be straighter in starting position")
263     status = "".join("Knees should be straighter in starting position")
264     status = "".join(improper_form)
265
266
267     elif state == "descending" or state == "ascending":
268         print("state: ", state)
269         if ((foot_distance - shoulder_distance) == 0.1 and
270             foot_angle > foot_angle_threshold):
271             print("foot distance: ", foot_distance)
272             print("foot angle: ", foot_angle)
273             status = "".join("Good posture during descent/ascent")
274     else:
275         improper_form = []
276         if ((foot_distance - shoulder_distance) == 0.1):
277             print("foot distance: ", foot_distance)
278             print ("Feet are too far apart.")
279             improper_form.append("Feet are too far apart.")
280         if foot_angle <= foot_angle_threshold:
281             print("foot angle: ", foot_angle)
282             print("Feet are not parallel.")
283             improper_form.append("Feet are not parallel.")
284         status = "".join(improper_form)
285
286     elif state == "bottom":
287         print("state: ", state)
288         if ((foot_distance - shoulder_distance) == 0.1 and
289             foot_angle > foot_angle_threshold and
290             knee_angle <= knee_angle_bottom_threshold and
291             hip_angle >= hip_angle_bottom_threshold):
292             print("foot distance: ", foot_distance)
293             print("foot angle: ", foot_angle)
```

Use math (angles, thresholds)
to determine appropriate state
and posture

Results

