Introducing Google AI Edge Portal (https://ai.google.dev/edge/ai-edge-portal): Benchmark Edge AI at scale. Sign-up (https://docs.google.com/forms/d/e/1FAIpQLSfTcGPycQve8TLAsfH46pBlXBZe9FrgJAClwbF7DeL1LgVn4Q/viewform) to request access during private preview.

Pose landmark detection guide for Python

The MediaPipe Pose Landmarker task lets you detect landmarks of human bodies in an image or video. You can use this task to identify key body locations, analyze posture, and categorize movements. This task uses machine learning (ML) models that work with single images or video. The task outputs body pose landmarks in image coordinates and in 3-dimensional world coordinates.

The code sample described in these instructions is available on GitHub (https://github.com/google-ai-edge/mediapipe-samples/tree/main/examples/pose_landmarker/python). For more information about the capabilities, models, and configuration options of this task, see the Overview (https://ai.google.dev/edge/mediapipe/solutions/vision/pose_landmarker/index).

Code example

The example code for Pose Landmarker provides a complete implementation of this task in Python for your reference. This code helps you test this task and get started on building your own pose landmarker. You can view, run, and edit the Pose Landmarker example code (https://colab.research.google.com/github/googlesamples/mediapipe/blob/main/examples/pose_landmarker/python/%5BMediaPipe_Python_Tasks%5D_Pose_Landmarker.ipynb) using just your web browser.

If you are implementing the Pose Landmarker for Raspberry Pi, refer to the Raspberry Pi example app (https://github.com/google-ai-edge/mediapipe-samples/tree/main/examples/pose_landmarker/raspberry_pi).

Setup

This section describes key steps for setting up your development environment and code projects specifically to use Pose Landmarker. For general information on setting up your development environment for using MediaPipe tasks, including platform version requirements, see the Setup guide for Python (/mediapipe/solutions/setup_python).

Attention: This MediaPipe Solutions Preview is an early release. Learn more (/edge/mediapipe/solutions/about#notice).

Packages

The MediaPipe Pose Landmarker task requires the mediapipe PyPI package. You can install and import these dependencies with the following:

```
$ python -m pip install mediapipe
```

Imports

Import the following classes to access the Pose Landmarker task functions:

```
import mediapipe as mp
from mediapipe.tasks import python
from mediapipe.tasks.python import vision
```

Model

The MediaPipe Pose Landmarker task requires a trained model that is compatible with this task. For more information on available trained models for Pose Landmarker, see the task overview Models section (https://ai.google.dev/edge/mediapipe/solutions/vision/pose_landmarker/index#models).

Select and download the model, and then store it in a local directory:

```
model_path = '/absolute/path/to/pose_landmarker.task'
```

Use the `BaseOptions` object `model_asset_path` parameter to specify the path of the model to use. For a code example, see the next section.

Create the task

The MediaPipe Pose Landmarker task uses the `create_from_options` function to set up the task. The `create_from_options` function accepts values for configuration options to handle. For

more information, see Configuration options (#configuration_options).

The following code demonstrates how to build and configure this task.

These samples also show the variations of the task construction for images, video files, and live stream.

Image (#image)Video Live stream (#live-stream)
(#video)

```
import mediapipe as mp

BaseOptions = mp.tasks.BaseOptions
PoseLandmarker = mp.tasks.vision.PoseLandmarker
PoseLandmarkerOptions = mp.tasks.vision.PoseLandmarkerOptions
VisionRunningMode = mp.tasks.vision.RunningMode

# Create a pose landmarker instance with the video mode:
options = PoseLandmarkerOptions(
    base_options=BaseOptions(model_asset_path=model_path),
    running_mode=VisionRunningMode.VIDEO)

with PoseLandmarker.create_from_options(options) as landmarker:
  # The landmarker is initialized. Use it here.
  # ...
```

Note: If you use the video mode or live stream mode, Pose Landmarker uses tracking to avoid triggering the model on every frame, which helps reduce latency.

For a complete example of creating a Pose Landmarker for use with an image, see the code example
(https://github.com/google-ai-edge/mediapipe-samples/tree/main/examples/pose_landmarker/python).

Configuration options

This task has the following configuration options for Python applications:

Option Name	Description	Value Range	Default Value
`running_mode`	Sets the running mode for the task. There are three modes: IMAGE: The mode for single image inputs. VIDEO: The mode for decoded frames of a video.	{`IMAGE`, `VIDEO`, `LIVE_STREAM`}	`IMAGE`

Option Name	Description	Value Range	Default Value
	LIVE_STREAM: The mode for a livestream of input data, such as from a camera. In this mode, resultListener must be called to set up a listener to receive results asynchronously.		
`num_poses`	The maximum number of poses that can be detected by the Pose Landmarker.	`Integer > 0`	`1`
`min_pose_detection_confidence`	The minimum confidence score for the pose detection to be considered successful.	`Float [0.0, 1.0]`	`0.5`
`min_pose_presence_confidence`	The minimum confidence score of pose presence score in the pose landmark detection.	`Float [0.0, 1.0]`	`0.5`
`min_tracking_confidence`	The minimum confidence score for the pose tracking to be considered successful.	`Float [0.0, 1.0]`	`0.5`
`output_segmentation_masks`	Whether Pose Landmarker outputs a segmentation mask for the detected pose.	`Boolean`	`False`
`result_callback`	Sets the result listener to receive the landmarker results asynchronously when Pose Landmarker is in the live stream mode. Can only be used when running mode is set to `LIVE_STREAM`	`Result Listener`	`N/A`

Prepare data

Prepare your input as an image file or a numpy array, then convert it to a `mediapipe.Image` object. If your input is a video file or live stream from a webcam, you can use an external library such as OpenCV (https://github.com/opencv/opencv) to load your input frames as numpy arrays.

Image (#image)Video Live stream (#live-stream)
(#video)

```
import mediapipe as mp

# Use OpenCV’s VideoCapture to load the input video.

# Load the frame rate of the video using OpenCV’s CV_CAP_PROP_FPS
# You’ll need it to calculate the timestamp for each frame.

# Loop through each frame in the video using VideoCapture#read()
```

```
# Convert the frame received from OpenCV to a MediaPipe's Image object.
mp_image = mp.Image(image_format=mp.ImageFormat.SRGB, data=numpy_frame_from_c
```

Run the task

The Pose Landmarker uses the **`detect`**, **`detect_for_video`** and **`detect_async`** functions to trigger inferences. For pose landmarking, this involves preprocessing input data and detecting poses in the image.

The following code demonstrates how to execute the processing with the task model.

Image (#image)VideoLive stream (#live-stream)
(#video)

```
# Perform pose landmarking on the provided single image.
# The pose landmarker must be created with the video mode.
pose_landmarker_result = landmarker.detect_for_video(mp_image, frame_timestar
```

Note the following:

- When running in the video mode or the live stream mode, also provide the Pose Landmarker task the timestamp of the input frame.
- When running in the image or the video model, the Pose Landmarker task blocks the current thread until it finishes processing the input image or frame.
- When running in the live stream mode, the Pose Landmarker task returns immediately and doesn't block the current thread. It will invoke the result listener with the detection result every time it finishes processing an input frame. If the detection function is called when the Pose Landmarker task is busy processing another frame, the task will ignore the new input frame.

For a complete example of running an Pose Landmarker on an image, see the code example (https://github.com/google-ai-edge/mediapipe-samples/tree/main/examples/pose_landmarker/python) for details.

Handle and display results

The Pose Landmarker returns a **`poseLandmarkerResult`** object for each detection run. The result object contains coordinates for each pose landmark.

The following shows an example of the output data from this task:

```
PoseLandmarkerResult:
  Landmarks:
    Landmark #0:
      x            : 0.638852
      y            : 0.671197
      z            : 0.129959
      visibility   : 0.9999997615814209
      presence     : 0.9999984502792358
    Landmark #1:
      x            : 0.634599
      y            : 0.536441
      z            : -0.06984
      visibility   : 0.999909
      presence     : 0.999958
    ... (33 landmarks per pose)
  WorldLandmarks:
    Landmark #0:
      x            : 0.067485
      y            : 0.031084
      z            : 0.055223
      visibility   : 0.9999997615814209
      presence     : 0.9999984502792358
    Landmark #1:
      x            : 0.063209
      y            : -0.00382
      z            : 0.020920
      visibility   : 0.999976
      presence     : 0.999998
    ... (33 world landmarks per pose)
  SegmentationMasks:
    ... (pictured below)
```

The output contains both normalized coordinates (**`Landmarks`**) and world coordinates (**`WorldLandmarks`**) for each landmark.

The output contains the following normalized coordinates (**`Landmarks`**):

- `x` and `y`: Landmark coordinates normalized between 0.0 and 1.0 by the image width (`x`) and height (`y`).
- `z`: The landmark depth, with the depth at the midpoint of the hips as the origin. The smaller the value, the closer the landmark is to the camera. The magnitude of z uses roughly the same scale as `x`.

- `visibility`: The likelihood of the landmark being visible within the image.

The output contains the following world coordinates (`WorldLandmarks`):

- `x`, `y`, and `z`: Real-world 3-dimensional coordinates in meters, with the midpoint of the hips as the origin.
- `visibility`: The likelihood of the landmark being visible within the image.

The following image shows a visualization of the task output:

The optional segmentation mask represents the likelihood of each pixel belonging to a detected person. The following image is a segmentation mask of the task output:

The Pose Landmarker example code demonstrates how to display the results returned from the task, see the code example (https://github.com/google-ai-edge/mediapipe-samples/tree/main/examples/pose_landmarker/python) for details.

Last updated 2025-01-13 UTC.