Table of Contents

Welcome

Project Willy

- History of Willy
- Project Willy
- Publicity
- Sponsors

Getting started

- Introduction to ROS
- Development Guide
- Driving Willy
- Manual
- Wiki Manual

Build of Willy

- Design history
- Hardware

Architecture

- Software Architecture
- ROS topic design

Raspberry Pi’s

- Sensor node
- Social Interaction node
- Power node

Components

- ROS master
- New ROS master on Lubuntu
- Sonar
- Lidar

1. Human detection

This component with a web cam gives Willy the ability to recognize upper human body parts with a OpenCV Haar Cascade. The algorithm tracks the outputted rectangles from the Haar Cascade output and counts the frames it has been seen and missed. Based on those figures it calculates an accuracy. When the accuracy drops below 30% or the rectangle has not been seen for 50 frames (that's about 2 seconds) it will be removed from the tracking. Due to the specific output of the human_detection it has its own ROS message type with parameters.

1.1. Repository

Windesheim-Willy/human_detection

1.2. General information

Package displaying debug output during development (see debug mode). You see the developer of the package raising 2 fingers telling he's two meters away from the webcam.

The packages can display the output to an window to help analyse the algorithm and allow you to see visuals whenever you want to do tweaks. The follow metrics are written on the screen;

- Number of humans currently detected
- Total number of humans detected during runtime
- The time it took to analyse the current frame (at 0.065 seconds that will make it about 15 fps)
- The ID of the human being tracked
- The frame dimensions (height and width)
- The accuracy of the result
- The distance in centimetres

To get this debug frame open `src/human_detection/src/main.h` and change the line

```
#DEFINE DEBUG FALSE
```

to

```
#DEFINE DEBUG TRUE
```

The result data within the red rectangle will also be published on a ROS channel with an custom message type.

1.3. Distance measurement

The distance of the result will be calculated based on the height of the rectangle as given by OpenCV. The formula is simple

```
    int distance = (300 - (this->br.y - this->tl.y)) * 2;
```

The calculation is performed in `src/human_detection/src/Tracking/Rectangle.cpp`. Where the variable **tl** means top left and the variable **br** means bottom right. The aim is a margin within 20 cm on both sides.

1.4. Human tracking

To make the algorithm function well it requires to run above 5 FPS. Since the tracking is done on the tl (top left) and br (bottom right) of a rectangle between each frame. The difference in position for each person may not too big thus requiring a high enough frame rate. Every frame a match has been made, the position of the rectangle is updated. Each match will be placed in an vector (array) as a Rectangle object. The equation can be found in `bool Rectangle::withinOffset(const Rect &rect)`.

The two classes *Rectangle* and *RectangleTracker* are used to track humans. When extending the tracker with a facial recognising the classes can easily be re-factored. The RectangleTracker can easily be a generic to support both types (assuming you'll write a new face class). The *OpenCVTracking* class handles all OpenCV output and gives the found *Rect* coordinates to to the *RectangleTracker* that matches them to an *Rectangle*.

1.5. Proposed enhancements

The Haar Cascades for the upper body work quite well, although within short distance facial recognition works far better. Most parts of the code can be reused for this purpose. The distance are quite accurate but are very volatile moving in their inaccuracy (+/-20 cm) limits. This can easily be fixed by making it an average and calculating it each tick (frame).

- Add face recognising
- Average distance
- Use AI models for recognising so the accuracy can be calculated by the model