Autonomous runway detection

--Project description –

Landing is one of the most critical phases during a flight. It's the second thing pilots learn in training after basic flight controls and recovery techniques.

One of the biggest fallacies out there is that a smooth landing is a good one. This isn't always true. Whilst it does make the ride more comfortable for our passengers, unless conditions are perfect and the runway is long, a soft landing can be a bad thing. Touchdown is the first opportunity we have to dump all the kinetic energy keeping the aircraft aloft.

If the touchdown is very soft, the airplane is still half-flying, meaning it still needs to be controlled carefully, taking more time to disperse the remaining energy through braking over a limited length of runway. It's also better for the wheels, as a smooth landing will scrape off the initial point of contact of the tires along the runway before friction spins the wheels up to the same speed as the aircraft.

Although, as aircrafts are able to land automatically, it's intended to be used in conditions of bad visibility (as in fog and mist). In fact, our wind limitations for landing are severely reduced if we need to carry out an automatic landing, so it's fortunate that it's very rarely foggy and blustery at the same time.

Figure 1 Turku airport runway during winter

Detection of runways in aerial images is part of a project to automatically map complex cultural areas such as a major commercial airport complex. This task is much more difficult than appears at first. Runways are not merely homogeneous strips in the image due to several markings on the surface, changes in the surface material and presence of other objects such as taxiways and aircraft.

Edge detection filter

Canny edge detection[1] is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. It has been widely applied in various computer vision systems. Canny has found that the requirements for the application of edge detection on diverse vision systems are relatively similar. Thus, an edge detection solution

[1] https://en.wikipedia.org/wiki/Canny_edge_detector

to address these requirements can be implemented in a wide range of situations. The general criteria for edge detection includes:

1. Detection of edge with low error rate, which means that the detection should accurately catch as many edges shown in the image as possible
2. The edge point detected from the operator should accurately localize on the center of the edge.
3. A given edge in the image should only be marked once, and where possible, image noise should not create false edges.

To satisfy these requirements Canny used the calculus of variations – a technique which finds the function which optimizes a given functional. The optimal function in Canny's detector is described by the sum of four exponential terms, but it can be approximated by the first derivative of a Gaussian.

Among the edge detection methods developed so far, Canny edge detection algorithm is one of the most strictly defined methods that provides good and reliable detection. Owing to its optimality to meet with the three criteria for edge detection and the simplicity of process for implementation, it became one of the most popular algorithms for edge detection.

Next generation runway detection

In the next generation of autonomous landings, costly computations is being transmitted to an off-aircraft cloud for high performance processing because of the limited power envelope of on-aircraft systems. This means that the aircraft must communicate with the ground continuously during the landing, and landing decisions are computed on ground HPC clusters, after which the results are transmitted back to the aircraft.

With such a procedure, it is essential to guarantee a strong communications link including a robust security channel to avoid unintended listeners and interception. Real-time requirements must always be guaranteed on both the on-aircraft systems and on the ground computer system. In this project we will focus on both of these aspects, and the students shall provide convincing solutions for this type of autonomous landing systems using runway edge detection filtering.

Project: IoT-Based Runway Detection for Autonomous Aircraft Landing

Overview

In this project we integrate a complete IoT system for assistance in autonomous landing of aircraft. We build a system capable of image filtering in real-time to spot the runway. This processed image is sent over a secure channel to a ground based cloud system for further analysis by a high performance backend to approve the safety or efficiency of the air craft positioning during the landing. In this project, the students are given various parts of program code, and the task is to integrate this code to form a working system.

Figure 2 Overview of the projectshows the overview of the system with data in form of picture frames to be analyzed by the edge detection algorithm to spot the runway. The edge detection algorithm is being executed inside a FreeRTOS system on-aircraft, and provides real-time image frames of the environment. In this project we provide these pictures in form of BMP images, while in a real-world scenario this would be provided by a camera connected to the real-time platform.

Once the picture frame is filtered, the frame matrix is sent to an encryption algorithm (also on-aircraft). This step provides the security for the frame to be transmitted to the ground cloud system. The secure frame is then forwarded to a TCP client, which transmits the frame to the cloud server. The cloud server interface is provided to the students in form of an IP address and its associated port number. The student must ensure that the filtered

frame is successfully transmitted to the cloud server and verify that the un-encrypted frame is identical to the frame before encryption.

Since this capstone is completely based on free software and requires no additional hardware (other than a PC), the FreeRTOS real-time OS is run in a PC simulator. This environment is identical to a real-platform solution, but timing measurements cannot be completely accurate since the real-time OS is executed in a host OS as a pthread. With this in mind, use the FreeRTOS timing measurement facilities to plan what would be needed if such a system was built in real hardware. Provide timings for:

- The edge detection algorithm with all provided BMP images
- The encryption algorithm for all the filtered frames
- The TCP connection to the cloud server

Additionally, the students shall investigate the timing requirements for transmitting the encrypted frames using a TCP connection via a wireless protocol of your choice to the cloud server. The student shall also provide a report for the confidence interval of TCP/IP transmission itself.

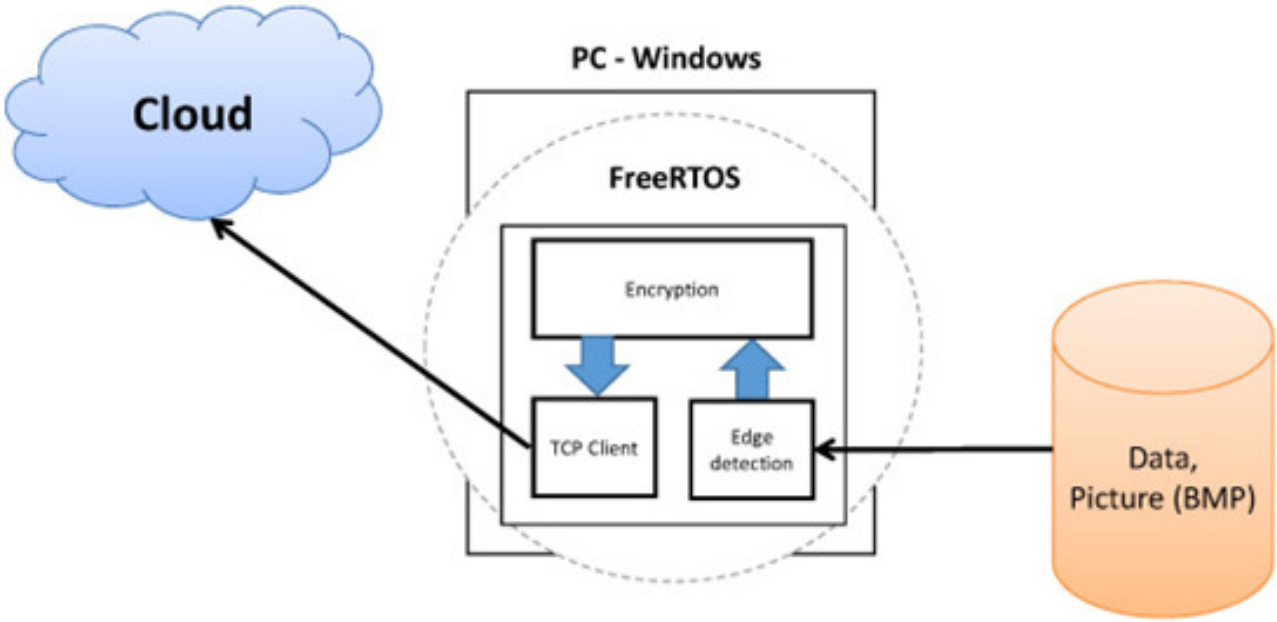

Figure 2 Overview of the project

Project parts

The project can be divided into four parts summarized as follows:

1 Integration of edge detection algorithm

The edge detection algorithm is capable of detecting sharp edges in a picture, which is used for outlining the runway and is used for the autonomous landing. Figure 3 shows an original (A) and a filtered image (B) from the perspective of the landing air craft. The code for this algorithm is given and the task is to integrate the functionality into FreeRTOS. The edge detection algorithm is provided on the Coursera webpage in the file “canny.c”.

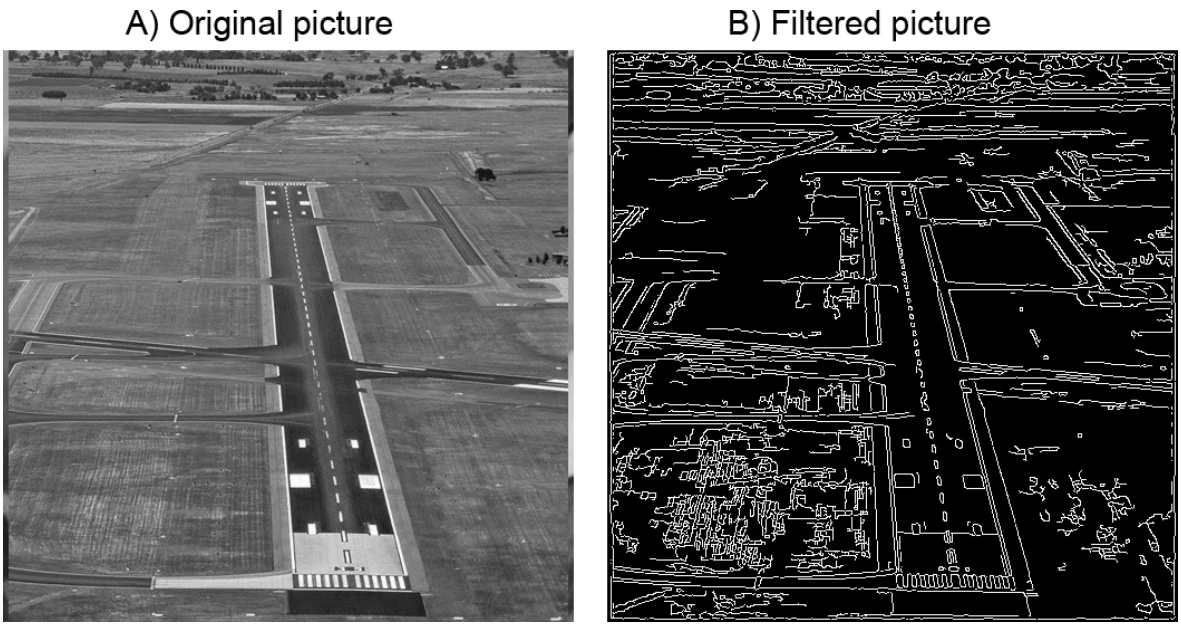

Figure 3 A) Original picture taken from the onboard camera B) Filtered image using edge detection

2 Integration of encryption algorithm

Since the filtered data contains sensitive information, the data is encrypted before sending it to the server. Figure 2 illustrates the encryption of the filtered image before it is transmitted to the cloud. The encryption code is given to the students and the task is to integrate this code into FreeRTOS. The encryption algorithm is provided on the Coursera webpage in the file “rsa.c”.

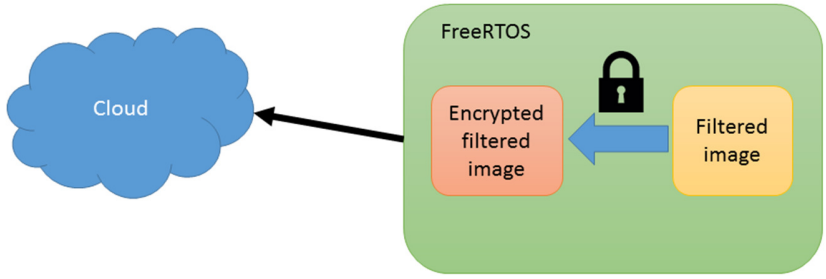

Figure 2 Encryption of filtered image

3 Transmission of data to cloud server

The last part of the integration is to transmit the encrypted data to the cloud server for storage. This shall preferably be implemented using a TCP client on the FreeRTOS side and a TCP server on the cloud server. The TCP client shall be able to transmit an encrypted video frame to the cloud server, and the server shall be able to receive the frame and decrypt the received frame. Please study the provided video lecture material to get an overview of how to implement the TCP client in FreeRTOS, also check out the FreeRTOS API for creating TCP connections.

The TCP server on the cloud side can be implemented in any language you feel comfortable with like C++, java, python or others. Note that you must properly manage the transmitted data in the correct format in order to be able to decrypt the frame. Demonstrate all steps in the written report and verify the decrypted frames.

Further analysis of the picture data on the cloud server side is not considered part of this project.

Instructions manual

As common practice for software engineering, a manual should be provided describing how to setup and execute the system. Since you are free to design and implement your system in many ways, there will be no "standard" system. Therefore it is required to include a how-to manual to get your system up and running. All parts of the system should be included in this manual including the TCP server and the source of the images to filter. It is therefore recommended that you develop the project with user friendliness in mind.

Feedback

You will receive feedback on your implementation from 3 different reviewers. During the implementation of the project it is highly recommended to attend the discussion forums on the Coursera page and the FreeRTOS forum at http://www.freertos.org/FreeRTOS_Support_Forum_Archive/freertos_support_forum_archive_index.html

After completing this project you will

- Gain insights in real industrial problems and learn methods used in every-day embedded systems
- Setup, implement and test a real-world embedded system
- Argue for design choices
- Implement communication interfaces between multiple actors in a large embedded system
- Estimate hardware and software requirements for a large scale system

Timeline

This project is estimated to require 3 weeks of work including preparations, reading documentation, watching videos, and the implementation itself.
The recommended timeline is estimated as follows:
2-3 days:

- Watching introduction week 1 videos
- Preparations
- Installation of software
- Studying algorithms and framework

Studying provided documentation
2-3 days:

- Watching week 2 videos
- Studying FreeRTOS API and testing some small examples
- Studying Canny filter and RSA implementation and test some small examples

15-17 days:

- Implement the project
- Evaluate the implementation
- Write final report

Good luck with the project!
-Simon & Farhoud