Internet of Things – Project Report

Creating a Cat Tracker Using the *TinyDuino* Platform

Thomas Reto Strub (strut1)
30 June 2017

Bern University of Applied Sciences
Engineering and Information Technology
Master of Science in Engineering

Contents

1 Project Goals

The basic idea of my project was to create GPS tracker to track where my cat wanders every day.

- I chose the *TinyDuino* platform, mainly for its size and weight. In order to attach the tracker to a small to medium sized cat, the device has to match the strict restrictions imposed.
- Because the *TinyDuino* platform only supports low-level radio, Bluetooth and WiFi boards, I decided to use the WiFi board. Neither do I want to deal with radio technology nor do I want the system to be limited to the Bluetooth range.
- I want the GPS tracker to be able to record the GPS positions of several hours along with their timestamps so they can be transmitted once the cat returns home. As a security measure, I decided to use an SD card for storage instead of the also available flash memory option, I was afraid the WiFi option would not work properly and I still wanted the device to work without.
- As a backup, I considered operating the device on a SD card only. This SD card would then have to be removed and the records accessed manually.
- If the WiFi option does work however, the device could send an additional status message along with the current GPS position. The Raspberry Pi can then use these status messages to display a home icon on its LED matrix to signify the cat is home.

2 Implementation Steps

- First, I assembled the boards and uploaded a first sketch (program).

Assembling the five boards was surprisingly easy. Most boards have the same size and they all have a standardised connection. For a first sketch, I decided to develop a simple Morse logic. This would be helpful later on because the device features no other way to print or show messages.

- I then studied existing libraries for the GPS board and decided to use one that includes parsing of NMEA messages.

I decided to use the *TinyGPS*[1] library instead of using both a GPS board library and a second one to parse the NMEA[2] messages. Early on, it became clear to me I did not want to parse these messages myself because there are a number of messages that all contain different data. To fetch all information necessary, a number of messages have to be parsed and the data has to be cached.

I also had to discover the *TinyGPS* library is very heavy on the extremely limited memory. Both the code and the variable memory is put under considerable stress and the compiler warns the sketch might become unstable and memory should be freed. If no floating-point numbers are used, the instructions to deal with them does not have to be loaded on the device, saving some code space.

- To store information to the SD card, there is a standard library already included.

Saving data to the SD card was comparably easy, however I discovered early on that SD card access is everything but reliable. Unfortunately, the sketch stopped working at this point and I realised I could no longer use the LED to send in Morse code. This conflict is not documented, however.

- At this stage, I tried to fix a few encountered problems.

Both the code and data memory consumption have been dangerously high. A warning has been emitted by the compiler warning of possible instable conditions if the static memory consumption could not be minimised. Interestingly, I had only declared three global variables. One Boolean flag and two library instances to read and parse the GPS records.

- The only currently available WiFi board comes with a supported standard library.

The current WiFi board is compatible with the *WiFi101*[3] standard Arduino library. While I was able to connect to a WPA2-secured network after configuring the board ports as instructed, I immediately realised at this stage I would never be able to successfully complete the project. The *WiFi101* library itself used over three quarters of the available code memory.

- Even though the project cannot be finished, I wanted to proof MQTT support could theoretically be implemented.

The chosen *Adafruit MQTT* library[4] is a lightweight, generic MQTT library for Arduino platforms. It is compatible with both standard-compliant Ethernet and WiFi libraries including *WiFi101*. It supports both subscribing and publishing to MQTT brokers. Instead of publishing GPS co-ordinates or status messages, I decided to publish a short message that will be displayed on the *Raspberry Pi*'s LED matrix utilising an MQTT queue and a *Node-RED* flow already created to display generic messages on the LED matrix earlier on when I had played with the *Raspberry Pi* earlier.

[1] https://github.com/florind/TinyGPS

[2] National Marine Electronics Association, organisation that created a widely used standard for GPS device messages

[3] Reference: https://www.arduino.cc/en/Reference/WiFi101, source code: https://github.com/arduino-libraries/WiFi101

[4] Source code: https://github.com/adafruit/Adafruit_MQTT_Library

3 Results

3.1 GPS Logger & SD Card Storage

The GPS / SD sketch is available at https://github.com/StrubT/IoTCatTracker.

The default sketch repeatedly reads the GPS co-ordinates, prints them to the serial port and stores them in CSV format to the SD card.

During the development phase, I introduced several pre-processor directives for configuration.

- The `GPS_DELAY_MS` directive controls the delay in milliseconds spent reading and decoding the GPS NMEA messages. Because I did not proceed to a point where I could actually test the battery lifetime, I did not have to optimise power consumption. To do so, a second delay could be introduced where the GPS module is switched off in between reading periods.
- Using the `GPS_FLOAT` flag, usage of floating-point numbers can be disabled in order to reduce code memory consumption. In turn, however, the GPS co-ordinates would then have to be post-processed afterwards.
- The `_GPS_NO_STATS` flag is emitted by the *TinyGPS* library and can be used to disable the calculation of GPS statistics.
- To minimise data memory consumption, the `GPS_MIN_INFO` flag can be enabled. In this case, only minimal GPS information is stored. Because of the already dangerously high consumption by global variables, however, it is unlikely even minimal GPS information could be stored in memory – rendering the SD storage unnecessary.
- The `DEBUG_DISABLE` flag allows serial port functionality to be disabled, allowing the code memory consumption to be reduced even more.
- The `LED_DISABLED` flag needs to be declared because – apparently – addressing the LED interferes with the SD board.
- During testing, the `SD_DISABLED` flag can be enabled to disable the SD storage. For unknown reasons, storing data to the SD card is unreliable and causing additional trouble when debugging.

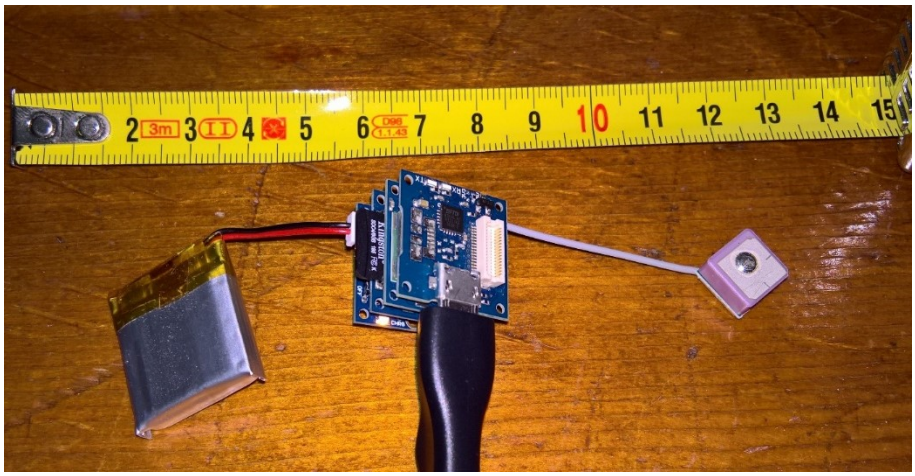

Figure 1: TinyDuino GPS / SD setup

3.2 WiFi Connectivity & MQTT Client

The WiFi / MQTT sketch is available at https://github.com/StrubT/IoTCatTracker/tree/feature/wifi.

This sketch simply tries to repeatedly connect to a predefined WiFi network and publish a MQTT message. There is not much to say about it; its implementation is pretty straightforward.

Unfortunately, the SIOT centre is offline once again, so I cannot include any pictures proving the setup is working properly.

3.3 GPS Record Visualiser

Because I did not manage to publish the GPS co-ordinates to the SIOT centre, I created a simple web page that can read the CSV file and display the path on a google map.

Figure 2: GPS record visualiser

4 Discussion

Although I did not fully achieve my goals and chose to skip properly testing my device on my cat, I completed two proofs of concept.

The GPS / SD setup reads GPS co-ordinates and stores them to an SD card. This setup could be used to track the cat – although the card would have to be manually removed and read on a computer. I also created a simple visualiser to read the CSV file.

The WiFi / MQTT setup successfully connects to a WiFi network and published an MQTT message to a now offline SIOT centre. This message is then displayed on the *Raspberry Pi's* LED matrix.

The major problem is that *TinyCircuit*, the manufacturer of the *TinyDuino* platform, recently introduced better components. The corresponding libraries consume significantly more code memory than the old libraries.