HACKING AND CONTROLLING TOY FLYERS
Abstract

The goal of this project is to demonstrate the feasibility, in the context development, of the implementation and use of cheap and ready-to-use commercial micro aerial vehicles (MAVs) as an acceptable alternative to full proprietary developed prototypes. Such development should prove to be interesting both in price, time and manpower. The implementation of a fully functional platform based on a selection of commercial MAVs will be done in order to demonstrate this statement. We will have at our disposition a PC, on which we will implement the MAV controlling software, an USB controller pad, which should be able to successfully communicate with the PC, and if possible, an embedded microcontroller, directly implanted into the retro-engineered MAV, in order to generate a control signal.
Contents

List of Figures ........................................ 3
List of Tables ........................................ 4
1. Introduction ........................................ 5
   1.1. Project background ............................... 5
   1.2. Motivations ..................................... 5
   1.3. Overview of past & current projects ............ 6
      1.3.1. Fixed-wing example: the MC2 ............... 6
      1.3.2. Rotary-wing example: the MAVSTAR Platform .... 7
      1.3.3. Quadcopter example: REELY platform .......... 8
      1.3.4. Flapping-wing example: DelFly Platform ....... 9
      1.3.5. The Commercial alternative .................. 9
2. Reverse Engineering .................................. 11
   2.1. Different strategies ............................ 11
   2.2. Snellight Alien Jump Jet ....................... 13
      2.2.1. Description ................................ 13
      2.2.2. Specifications .............................. 13
      2.2.3. Retro-engineering the AJJ ................... 15
         2.2.3.1. Preliminary Notions ..................... 15
         2.2.3.2. The handset ............................. 16
         2.2.3.3. The Flyer ............................... 16
         2.2.3.4. Signal description ....................... 16
   2.3. Sanhuan Copter MaxZ Swift 6020-1 ............... 18
      2.3.1. Description ................................ 18
      2.3.2. Specifications .............................. 19
      2.3.3. Retro-engineering the MaxZ helicopter ....... 19
         2.3.3.1. Preliminary Notions ..................... 19
         2.3.3.2. The handset ............................. 20
         2.3.3.3. The Flyer ............................... 20
         2.3.3.4. Signal description ....................... 20
3. Interface implementation .............................. 24
   3.1. Computer interface description .................. 24
      3.1.1. Cahier-des-charges ......................... 24
         3.1.1.1. Inputs .................................. 24
         3.1.1.2. Outputs ................................. 24
      3.1.2. Classes description .......................... 25
         3.1.2.1. MAVPCI.CPP ............................. 26
         3.1.2.2. MAPPING.CPP ............................ 26
         3.1.2.3. HANDSETINPUT.CPP ...................... 27
         3.1.2.4. SIGNAL.CPP ............................. 28
         3.1.2.5. COM.CPP ................................. 28
      3.1.3. Libraries/Compiler constraints ............... 29
   3.2. Embedded Firmware interface description .......... 29
      3.2.1. Cahier-des-charges ......................... 29
         3.2.1.1. Inputs .................................. 30
         3.2.1.2. Outputs ................................. 30
      3.2.2. Class description ............................ 30
4. Final results ........................................ 33
   4.1. Alien Jump Jet .................................. 33
   4.2. MaxZ Helicopter ................................ 33
5. Conclusion ........................................... 35
   5.1. Discussion ..................................... 35
## Contents

5.2. Future work .......................... 35
5.3. Acknowledgments .................. 35

References .............................. 37
  WEBSITES ............................. 37
  BOOKS / PUBLICATIONS ............... 37

Appendix ................................. 38
  Appendix 1 : MAVPCI user guide ....... 38
  Appendix 2 : USB-BT-RS485 Converter description .......... 42
  Appendix 3 : AJJ Photodiode description .......... 50
List of Figures

1.3.1 The LIS's MC2 Fixed wing microflyer ........................................ 6
1.3.2 MC2 Description ........................................................................... 7
1.3.3 The MAVSTAR rotary-wing flyer .................................................. 8
1.3.4 Concept view of The REELY quadcopter ...................................... 8
1.3.5 DelfFly Micro. Flapping-wing micro flyer .................................... 9
1.3.6 From left to right: Sneight Alien jump jet, Swift R/C MaxZ Copter,
ParkZone Vapor, WooWee Flytech Dragonfly, Silverlit X251 jet ... 9

2.1.1 Hacking and controlling toy flyers ................................................. 11
2.1.2 Simplified illustration of the a strategy 1 example. Controlling directly
the motors of the flyer. Our own PCB is in pink and communicates
through the communication channel (COM) with the computer (not
represented here) using Bluetooth ..................................................... 12
2.1.3 Simplified illustration of a strategy 2 example: Emulating the
communication with the handset from our own PCB (pink) connected
through a communication channel (COM) to our PC platform .... 12
2.1.4 Simplified illustration of a strategy 3 example. Emulating the joystick
impulses from our own microcontroller and injecting our signal into
the handset microcontroller. In green is the commercial handset provided
with the flyer .................................................................................. 13
2.2.1 AJJ Axes of motion specification (throttle is not shown here) ........... 14
2.2.2 AJJ handset modes specification, as described in the user manual .... 14
2.2.3 AJJ Handset controller without front cover and joysticks cover ..... 15
2.2.4 AJJ signal description. The parameters are defined by the width between
the positive pulses. This is the signal in 'idle' (while not moving the
joysticks). .................................................................................. 16
2.2.5 AJJ control signal over the first parameter period: roll, while the roll
joystick is in idle ........................................................................... 17
2.2.6 AJJ control signal over the first period: roll, while the joystick is fully
pushed in the right direction .......................................................... 18
2.2.7 AJJ control signal over the first period: roll, while the joystick is fully
pushed in the left direction ............................................................ 18
2.3.1 Tentative of retro-engineering the handset of the MaxZ Copter ....... 20
2.3.2 MaxZ Copter Photodiode visible to the left. The pins description is the
following : Control(1), Vcc(2), Ground(3) ................................... 21
2.3.3 MaxZ Copter : Illustration of a a '000011111' to demonstrate
the 0-to-1 transition .................................................................... 21
2.3.4 MaxZ Copter signal description summary. When we set the throttle
joystick to a very small value, all other parameters are left to 0 ....... 22
2.3.5 MaxZ Copter maximum throttle signal. When reading the first 8 bits we
get the value '01111000'. ............................................................. 23
2.3.6 MaxZ Copter maximum throttle signal. When adding a maximum yaw
bias in the right direction (rolling the switch down in the handset). The
value we read here for the bias is '1111010'. ................................. 23
2.3.7 MaxZ Copter maximum throttle signal. When adding a maximum yaw
bias in the left direction (rolling the switch up in the handset). The value
we read here for the bias is '011100'. ............................................... 23
3.2.1 Summary of the AJJ.c Finite Machine states transition. States are in red.
Each state corresponds to an output of 0 or 1. The blue dotted lines are
the «events» times in the FSM ......................................................... 32
4.1.1 Final result : Alien jump jet connected to the PC platform through the
embedded microchip ................................................................... 34
List of Tables

1.1 Summary of the selection of commercial MAVs available for the project [12] 10
2.1 Summary of the Alien Jump Jet control signal construction. . . . . . . . . 17
1. Introduction

1.1. Project background

During the past couple of years, there has been a large outburst of research in the field of Unmanned Aerial Vehicles (UAV). Adding the third dimension in robotic development widely increased the possibilities of unmanned vehicles and opened new areas in robotics and automation. The development being driven by commercial, research, government, and military purposes. The Micro Aerial Vehicles (MAV) is one of these current and constantly evolving areas of research, driven of course by the race to automation, but also by the one of miniaturization. These MAVs\(^1\) are acting as full research platforms for the development and validation of novel hardware, control algorithms and paradigms including vision systems, autonomous steering algorithms and swarm intelligence. This is why it would be of great benefit to have large amounts of these MAVs immediately available, both for research as well as for education.

1.2. Motivations

As proven by many of the research performed here within EPFL and amongst other school as well, the development of novel MAVs can prove to be very expensive and time consuming. As the results often show to be richly rewarding and enlightening for those involved, low level hardware development can request a lot of manpower in a wide panel of different fields, from electronics to aeronautics through informatics, putting a lot of financial weight on the research. Furthermore since the development of these MAVs is typically not optimized for mass production mostly only prototypes exist, limiting their operational use for further testing. Fortunately we nowadays observe that more and more MAVs can be found on the toy market, offering a viable and cheap alternative. Although they come at a much cheaper price than their research counterparts, these commercial MAVs are not exempt of drawbacks. The first and foremost being that they were not optimized to be fully controlled in a laboratory environment, as they were developed for the general public. Therefore some more acute functionality were voluntarily restrained or even gave up by the developers. Consequently, further investment has to be made to study the possibility to gain full control of these platforms, which in time could prove to be a very efficient way of implementing testing and improving current or future autonomous flying paradigms, such as obstacles avoidance or distributed meteorological information collecting. The increased supply of these MAVs compared to a limited number of prototypes could prove to be very useful in a "test-and-improve" development pattern, as the drive in the research progress.

The goal of this research is to show the feasibility of developing an usable MAV research platform based on already fully functional commercial ones. In order to do this, we first have to get an overview of what has been done in this area of research to the current day. Then in the given enlightenment, we will carefully select some cheap and accessible commercial flyers on which we will work for the rest of this project. This will consist of taking control of these platforms by the simpler means possible. We will try to fully control them using

\(^1\) Also called "Flyers" by missomer.
1. Introduction

Figure 1.3.1. The LIS’s MC2 Fixed wing microflyer.

a standard Win32/64 computer platform connected to a gamepad, and communicate with the flyers using our own communication channel USB/Bluetooth. We will try to integrate a micro-controller into the MAV which will act as a reception and processing platform, converting our control signal into spatial movement. This micro-controller should be easily upgradable in view of future improvements in the robotics field. The understanding of underlying MAVs operation is crucial, as it can prove to be much useful for further research.

1.3. Overview of past & current projects

As stated in the Motivations section, various studies have been performed, each using different micro aerial vehicles designs depending on the needs of the research performed. Micro air vehicles are either fixed-wing aircrafts, rotary-wing aircrafts (helicopters), or flapping-wing (of which the ornithopter is a subset) designs; with each being used for different purposes. Fixed-wing aircraft require higher, forward flight speeds to stay airborne, and are therefore able to cover longer distances; however they are unable to fully effectively maneuver inside structures such as buildings. Rotary-wing designs allow the craft to hover and move in any direction, at the cost of requiring closer proximity for launch and recovery [1]. They can come in varying rotors configurations, from standard main rotor/tail rotor helicopter configuration to double triple or quadruple rotor configurations (quadcopter), each having their own advantages and drawbacks. Let’s start with a selection of fully developed research micro air vehicles, then we will have a quick overview of what could be their commercial alternative in term of avionics.

1.3.1. Fixed-wing example : the MC2

The MC2 is a 10-gram, fixed wing, autonomous microflyer developed here at EPFL by the Laboratory of Intelligent Systems (LIS). As stated on their website, the purpose of this ongoing experiment is to demonstrate autonomous steering of a 10-gram microflyer (the MC2) in a square room with different kind of textures on the walls [2]. The MC2 (Figure 1.3.2) is based on a MicroCeline platform2. It integrates a 5-gram living room flyer produced by Didel equipped with a 4mm geared motor (a) and two magnet-in-a-coil actuators (b) controlling the rudder and the elevator (b). When fitted with the required electronics for autonomous navigation, the total weight reaches 10 grams.

The custom electronics consists of a microcontroller board (c) featuring a PIC18LF4620 running at 32MHz, a Bluetooth radio module (for parameter

---

2 http://www.didel.com/07mceline/indexF.html
1. Introduction

1.3.2. Rotary-wing example: the MAVSTAR Platform

The MAVSTAR platform is a coaxial helicopter developed at The University of New South Wales (UNSW) - Australia. The flyer is approximately 30cm in diameter (Figure 1.3.3). The flight time and gross weight are 18mins and 489g respectively when using a 1320mAh battery. A weighted flybar is attached to the top blades to mechanically stabilize the MAV. The MAV utilizes the most advanced technology and materials, including custom-made carbon fiber blades and frames, brush-less motors and a Li-Po battery. There are several on-board sensors for semi-autonomous control, such as an ultrasonic range finder, a video camera and an inertial measurement unit (IMU). The videos (indoor and outdoor tests) show the developed MAV's ability to hover and fly stably in indoor and outdoor environments.

Started by Dr Tomonari Furukawa in late 2006, the project was developed primarily throughout 2007. It resulted in the design and construction of a team of MAVs and unmanned ground vehicles (UGV) which participated in the MAV08 competition in March 2008 held in Agra, India. The team won the award for ‘Best UGV Performance’. In early 2009 the team worked towards the IMAV09 competition, held in Pensacola, Florida in June 2009. At that event, the team won third place in the indoor flight competition. Currently the team is working hard to develop a brand new 3rd generation MAV platform to be

Figure 1.3.2. MC2 Description

monitoring), and two camera modules, which comprise a CMOS linear camera (TSL3301) and a MEMS rate gyros (ADXRS150) each. One of those camera modules (d) is oriented forward with its rate gyro measuring yaw rotations, and is mainly used for obstacle avoidance. The second camera module (c) is oriented downwards, looking longitudinally at the ground, while its rate gyro measures rotation around the pitch axis. Each of the cameras have 102 gray-level pixels spanning a total field of view of 120°. In order to measure its airspeed, the MC2 is also equipped with an anemometer (e) consisting of a free propeller and a hall-effect sensor. This anemometer is placed in a region that is not blown by the main propeller (a). The 65mAh Lithium-polymer battery (f) ensures an autonomy of approximately 10 minutes. A program is currently under development, which will allow for trajectory reconstruction.

Figure 1.3.2. MC2 Description
1. Introduction

Figure 1.3.3. The MAVSTAR rotary-wing flyer

Figure 1.3.4. Concept view of The REELY quadcopter

launched at the end of 2010. This team will compete using the new platform at next year’s international MAV event [3].

1.3.3. Quadcopter example : REELY platform

The REELY platform is a very interesting & original autonomous quadcopter (VTOL) developed at ETH Zurich, in the Autonomous Systems Lab (ASL) [11]. The goal is to combine small aerial vehicle technology with an entertainment vision. They develop a flying entertainment robot based on quadrotor technology. The robot is designed as a 775 mm diameter flying film reel with the ability to show movies on small 4.3" OLED displays arranged around the reel. Once a trajectory is defined, the robot fly its way autonomously and afterward find back to the docking station for recharging. Reely has a lightweight structure made of carbon and glass fibers giving the MAV a total weight of 1550 g. It is equipped with powerful brushless outrunner e-motors and lithium-polymer accumulator technology. Pictures and films can be shown on the six OLED displays all around the reel. A specially developed trajectory planning software allows Reely to fly given routes automatically without the assistance of a pilot.

At the moment Reely is in the final production phase. The structure will be completed in the next days. Software development and implementation is on its way. The maiden flight of Reely is expected to take place within the next four to five weeks\(^3\). The project was developed alongside the MuFly project in the same lab. Reely is very fascinating in the sense that it is a perfect demonstration of the wide Micro Aerial Vehicle applications possible.

\(^3\) It seems that by the end of the making of this report, the Reely’s team were able to make their maiden flight. Check [http://www.reely.ethz.ch/index.html](http://www.reely.ethz.ch/index.html) to see the recently posted video!
1.3.4. Flapping-wing example: DelFly Platform

As an interesting flyer design the flapping-wing MAVs are nonetheless amongst the most complicated designs. Fully stable and functional flyers require a meticulous development phase. Indeed, these reaches the limits of our understanding of aerodynamics, since the simulation of ornithopter flight is still very problematic.

With such a design, DelFly is a Flapping-wing micro air vehicle platform currently developed at the Technical University of Delft, Netherlands. The development successfully resulted in the DelFly 1 with a size of 50cm and the DelFly 2 with a size of 28cm, both in 2005. It currently continues with the DelFly Micro. The DelFly micro is a 10 cm, from wingtip to wingtip micro flapper, weighting only 3.07 grams with an additional 1 gram for the embarked 30 mAh lithium polymer, giving it three minutes flight. The wings flapping frequency is 30 Hz and the range is 50 meters. As stated in the official developers website, it embarks a 0.4 gram microcamera essential for improving the autonomy of small flying robots, allowing experimentation with new ideas from fields such as computer vision and artificial intelligence. As stated in the official website, the final goal of this experiment is to achieve flight without the intervention of a pilot [4].

1.3.5. The Commercial alternative

Now that we have an overview on some past and current studies on the micro air vehicle field, let’s try to have a look of what could be the commercial alternative to the fully developed projects mentioned in the previous section. The table 1.1 provides a small summary of some interesting toy flyers to hack for this project.

First we have the Sneflight’s Alien jump jet which has the same quadcopter configuration than the Reely platform developed at ETH cited above. This sim-

---

4 Will Regan, Floris van Breugel, Hod Lipson. 2006 - Towards Evolvable Hovering Flight on a Physical Ornithopter - Computational Synthesis Lab Cornell University, Ithaca NY 14853, USA
1. Introduction

<table>
<thead>
<tr>
<th>MAV NAME</th>
<th>LENGTH/ WIDTH (ROTOR DIAMETER)</th>
<th>WEIGHT</th>
<th>BATTERY</th>
<th>CONTROLS</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sneligh Alien Jump Jet</td>
<td>360mm/340 mm</td>
<td>65g</td>
<td>140mAh</td>
<td>3-Axis + throttle, 4 channel infra-red</td>
<td>CHF 130.-</td>
</tr>
<tr>
<td>Sanhuan Swift MaxZ R/C Copter</td>
<td>195mm / 45mm / 173mm</td>
<td>41g</td>
<td>130mAh</td>
<td>2-Axis+ throttle, 3 channel infrared</td>
<td>CHF 50.-</td>
</tr>
<tr>
<td>ParkZone Vapor</td>
<td>387mm/375mm</td>
<td>15g</td>
<td>70mAh</td>
<td>2 Axis + throttle, 3 channel 2.4GHz radio</td>
<td>CHF 160.-</td>
</tr>
<tr>
<td>Silverlit Xtwin Jet</td>
<td>310mm/250mm</td>
<td>20g</td>
<td>130mAh</td>
<td>1 Axis + throttle, 2 channel radio</td>
<td>CHF 50.-</td>
</tr>
<tr>
<td>WooWee Flytech Dragonfly</td>
<td>420 mm/ 28 g</td>
<td>140mAh</td>
<td>LiPo</td>
<td>1 Axis + throttle, 2 Channel Radio</td>
<td>CHF 60.-</td>
</tr>
</tbody>
</table>

Table 1.1. Summary of the selection of commercial MAVs available for the project [12]

Similar avionics give us an idea of what we could be capable of doing if we were to take control of the Alien Jump Jet. Second we have Sanhuan MaxZ Copter who would correspond in terms of avionics, to the Mavstar alike platforms developed through research. And then we have the ParkZone Vapor who is very similar to the MC2 platform from the LIS, and the Flytech dragonfly which is a flapper, exactly like the DelFly platform. These flyers are all available at a relatively affordable price, between CHF 50 for the Maxz helicopter to a CHF 160 for the fixed-wing Parkzone Vapor, as shown in table 1.1.

As we see, there now exist a lot of different commercial toy flyers developed these last years, who pretty much look like some of the outcomes of former studies on the MAV field. For this project, we choose to take in priority control of the Snelight’s Alien Jump Jet, as it is amongst the most interesting toy flyers of the lot when speaking of the potential improvements we could implement on such a platform. The AJJ is also the most agile, as you can control its movements amongst 3 different axis (pitch, yaw & roll) in addition to throttle. Once we have developed a working platform with the AJJ, it shouldn’t be to difficult to extend & adapt the results & the working software code to other toy flyers. Therefore we will first focus on the Alien Jump Jet & then on others, starting with the Sanhuan MaxZ Copter, which is one of the most stable (in term of flying capacity) platform of the lot, making it also very interesting to take control of.
2. Reverse Engineering

2.1. Different strategies

We want to take control of these different platforms by using, if possible, the simplest and most efficient ways available. In order to show the feasibility of such takeover, we will have at our disposition a PC on which we will implement the MAV controlling software, an USB controller pad which should be able to successfully communicate with the PC, and if possible, an embedded microcontroller, directly implanted into the retro-engineered MAV (although will not always be the best solution, as shown in this section). The goal of the microcontroller will be to receive our own control signal generated in the PC interface, and use it to emulate the original electronic control signal(s). That would give us a flyer fully controllable through the PC interface while using our own communication channel. The embedded microcontroller should then allow anyone to upgrade the functionality of this platform in order to suit their needs. The figure 2.1.1 illustrates these goals. As we consider “hacking” through commercial MAVs, different strategies appear to be applicable, depending on the configuration of their embedded electronics. Indeed, we need to know exactly how the original control signal is sent from the original handset to the MAV original microcontroller in order to exactly know how/where to inject our signal into the system. These strategies are stated below:

An early strategy would consist of directly cutting off the original embedded electronics, and replace it by our own (Fig 2.1.2). We would then precisely generate the right signal to each motor in our own microcontroller, in order to fully control the flyer. The advantages of such strategy is that it is the easiest one in terms of retro-engineering and can therefore be quickly implemented. This strategy is also the more economic in terms of payload concern for the MAV. However, this strategy has a big flaw in the sense that it removes all the advantages of taking control of already tried and tested MAVs. Indeed, by removing the original electronics, we also remove some of their key features, like the gyroscopic stabilization system, which is essential for stability. Applying such strategy would therefore mean having to face major avionics problems again, which is the same as designing our own flyer from scratch. Therefore this strategy should be considered as last resort, when all others have failed.

Another more efficient strategy would consist of cutting off the reception photodiode embedded in the MAV and replace it by our own embedded PCB (Fig 2.1.3). Our microcontroller would then generate the control signal as if it was the photodiode, and send it directly to the original embedded microcontroller. The advantages of such strategy is that unlike the first strategy, it allows us to keep all the benefits of the initial commercial platform on which we are working, the first of them being a simple and stable control system. However,
such a strategy suppose that we are able to fully generate the correct signal, and thus presupposes that we are completely aware of the communication signal structure. If it turned out that the communication signal between the original handset and the flyer’s original embedded microcontroller is a coded signal, then applying this strategy could prove to be very tough as it would mean that we have to break the code in order to generate a correct signal. In term of payload concerns, the efficiency of this strategy depends only on the capacity to miniaturize our PCB, which should not be a problem. This is the prioritized strategy, as it is the better compromise between payload concerns and efficiency.

A third strategy would consist of emulating the joystick impulses in our own microcontroller and inject them directly to the original handset microcontroller. There would be encoded like regular joystick moves, and send to the MAV using the regular pipeline. Benefits of such a strategy lie in the fact that we don’t have to decode the communication signal between the original handset and the flyer. This strategy would therefore be effective in the case that we are not able to fully decode the communication channel between the flyer and it’s original handset. The big flaw here lies in the fact that if we want to embark our PCB in the MAV and use our own communication channel with between the flyer and the PC interface, the flyer would have to carry the original handset PCB plus our own PCB as well with the original embedded electronics, thus leading to a serious payload concern. Indeed even when we fully extract the official handset PCB form it’s original plastic case, it remains quite large, 10cm over 3cm for the Alien Jump jet, and heavy, around 200g with the four AA batteries also for the AJJ, as it was not originally designed to be compact and mobile.
2. Reverse Engineering

2.1. Simplified illustration of a strategy 3 example. Emulating the joystick impulses from our own microcontroller and injecting our signal into the handset microcontroller. In green is the commercial handset provided with the flyer.

2.2. Snellflight Alien Jump Jet

2.2.1. Description

The Alien Jump Jet is a genuine VTOL ready-to-fly miniature jump jet, inspired by the famous vertical takeoff Harrier jets. The most interesting fact about this flyer is that it can take off, hover, and land like a helicopter while flying around like a jet. Of the commercial models described earlier on this report, it is the one having the more spacial movement liberty axis (Fig. 2.2.1), as you can control the yaw the roll & the pitch as well as the throttle independently. It is powered by four gyroscopically balanced rotors for precision omnidirectional flying, giving the AJJ incredible maneuverability while almost not scarifying stability. When you control it you can select to have either a mode 1 or mode 2 transmitter. Mode 1 has throttle on the right stick, mode 2 has throttle on the left stick (Fig 2.2.2). It comes with the aircraft, replacement blades, a transmitter and a wall socket charger with an adapter for most major countries.

It is controlled by a 4 channel infra-red handset, but a radio controlled upgrade can be found on the Internet. The IR signal is very powerful and it works well in the home, but if you want to fly it outdoors or in large open areas you will want to upgrade it to RF [5].

2.2.2. Specifications

- Product model number: NO.EF200808
- Manufacturer: Snellflight Ltd [5]
- Motors: 4
- Weight: 65g
- Wingspan: 34cm
- Battery: 3S 140mAh Li Po
- Flight Time: Around 4 - 5 minutes
- Control: Proportional 3-axis 4-channel Infra-red (THROTTLE, YAW, PITCH, ROLL)
- Stabilization: 3-axis auto-zeroing computer gyroscopes
- Power Input: 100v to 240v AC - 5 watts
- Handset: 4CH IR strong plastic with 2 different joystick configuration modes (Fig 2.2.2)
- Included: 1 Aircraft, 1 Handset (mode 1, mode 2), 1 Charger
Figure 2.2.1. AJJ Axes of motion specification (throttle is not shown here)

Figure 2.2.2. AJJ handset modes specification, as described in the user manual.
2.2.3. Retro-engineering the AJJ

2.2.3.1. Preliminary Notions

The handset has four controls operated by moving the two joysticks either up and down or side to side. Moving one of the joysticks causes the aircraft’s propellers to change speed in the correct combinations to move the aircraft in the appropriate direction. For example, raising the throttle causes all four propellers to speed up simultaneously so as to lift the aircraft vertically upward. Pushing the Elevator stick forwards causes the front propeller to slow down, and the rear one to speed up. This makes the aircraft tilt forwards as if to dive.

All of the controls except for the throttle, cause the aircraft to rotate about a particular axis, altering its orientation in the air. The Jump Jet has three such rotational, or attitude controls, corresponding to the three primary axes of motion, called pitch (tilting the nose up and down), roll (raising and lowering the wingtips) and yaw (horizontal steering, like a car). See figure 2.2.1 for the axis description. In order to make the aircraft fly in a particular direction, the pilot must tilt the aircraft correspondingly; for example to fly forwards, the pilot needs to pitch the aircraft forwards (nose down).

It is very important to understand that the aircraft will not immediately return to the horizontal state and stop moving when the axis joysticks are centered. The neutral positions simply mean “leave the aircraft attitude in the current state”. Therefore If a rapid stop is needed, the pilot must actively apply control in the opposite direction.

In order to help stabilize the aircraft, the propellers receive speed adjustments from the three on-board motion sensors (gyros). Each gyro sensor detects movement in one of the axes of motion, pitch, roll or yaw. When a movement is detected, the gyro drives the propellers so as to try to counteract the motion as it occurs. This prevents the aircraft from exponentially heaving in any direction when a control is actioned. This makes the aircraft much easier to fly than it would be otherwise. The gyros are very sensitive, and they need to self calibrate before flight by setting their zero movement references whilst the aircraft is stationary on the ground. This happens during the first second after switching on the handset. It is therefore very important not to move the aircraft during the calibrating time.

2.2.3.2. The handset

Since we want to inject our own control signal into the flyer’s electronics, we first need to know how to construct it. We need to have the purest possible output control signal as a source for measurements. These measurements will later allow us to implement our own control signal with the less error possible. Therefore, as the first move it would be natural to try to collect the information from the source, the handset.

Unfortunately the signal is directly encoded into analog format, and sent
to the infra-red emitting LEDs inside the embedded SAM88RCRI\(^1\) Samsung microcontroller. It could be possible to implement the 3rd strategy, however the approach favored is that we first want to control if the signal is not coded in order to implement the first strategy. Let's now look on the flyer side to see what we can come up with.

### 2.2.3.3. The Flyer

When we dismantle the plastic shell of the Alien jump jet we can clearly see the communication photodiode used to receive the infra-red signal. We can also see the PCB, the various chips, and more importantly the photodiode module serial number: PC3388. Using this model number to search over Internet, provided us the complete module description, including the functioning voltages and the modulating/demodulating patterns\(^2\). The photodiode is connected to the embedded microcontroller through three pins, one for the ground, one for the power supply, and one for the control signal. By connecting our scope directly through the photodiode, between the ground connection and the control signal connection, we were able to get the signal described in the results section. Here it is important to remember the mode used in the original controller in order to not mix the parameters as we measure them. The following result were get while using mode 2 transmission.

#### 2.2.3.4. signal description

![AJJ signal description](image)

Figure 2.2.4. AJJ signal description. The parameters are defined by the width between the positive pulses. This is the signal in ‘idle’ (while not moving the joysticks).

Luckily, the signal between the photodiode receptor and the microcontroller is not coded (Fig 2.2.4). It consists of five constant positive width pulses, each separated by non constant negative width. The signal is consistent, in the sense that as long as the flyer is turned on, the signal is transmitted between the flyer & the handset. By moving the parameter and analyzing the results we were able to get the following results: The roll is controlled by the width of the first gap (a), the pitch is controlled by the width of the second gap (b), the throttle is controlled by the width of the third gap (c) and the yaw is controlled by the width of the last gap (d). The complete period (the pulse length followed by gap length) for each parameter varies between 1ms to 2ms, depending on

---

1. S3C9452/C9454 20-pin DIP/SOP Package
2. Description of the PC3388 IR Photodiode is included in the Appendix 3 section of this report.
2. Reverse Engineering

<table>
<thead>
<tr>
<th>PARAMETER CONTROL</th>
<th>PARAMETER NEGATIVE WIDTH</th>
<th>NEGATIVE WIDTH VALUE when the joystick is pushed left/down to the max.</th>
<th>NEGATIVE WIDTH VALUE when the joystick is in idle position</th>
<th>NEGATIVE WIDTH VALUE when the joystick is pushed right/up to the max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>THROTTLE LEFT JOYSTICK Y AXIS</td>
<td>&lt;c&gt;</td>
<td>-</td>
<td>1720</td>
<td>720</td>
</tr>
<tr>
<td>YAW LEFT JOYSTICK X AXIS</td>
<td>&lt;d&gt;</td>
<td>720</td>
<td>1220</td>
<td>1720</td>
</tr>
<tr>
<td>PITCH RIGHT JOYSTICK Y AXIS</td>
<td>&lt;b&gt;</td>
<td>1720</td>
<td>1220</td>
<td>720</td>
</tr>
<tr>
<td>ROLL RIGHT JOYSTICK X AXIS</td>
<td>&lt;a&gt;</td>
<td>720</td>
<td>1220</td>
<td>1720</td>
</tr>
</tbody>
</table>

Table 2.1. Summary of the Alien Jump Jet control signal construction.

When we carefully look on each parameter to see it’s behavior, we see that the width of each parameter’s gap is linearly correlated with the handset joystick positions. Indeed, if we take the roll, we see that slowly pushing the roll joystick in the right direction linearly increase the complete first period width from 1.5ms, which is the idle median value (Fig 2.2.5) to 2ms (Fig 2.2.6) which is the maximum roll value you can give in the right direction. Slowly pushing the roll joystick in the left direction linearly reduce the complete roll parameter period width to 1ms (Fig 2.2.7). The yaw parameter period behaves exactly the same as the roll parameter period, as stated in the summary table (Table 2.1). For the pitch parameter period, the behavior is very similar but there is a small difference: Slowly pushing the pitch joystick up will linearly decrease the pitch parameter period from the idle value of 1.5ms to 1ms, while pushing it downwards will linearly increase this period to 2ms. As for the throttle parameter, it is slightly different. The throttle parameter period (pulse length...
2. Reverse Engineering

Figure 2.2.6. AJJ control signal over the first period: roll, while the joystick is fully pushed in the right direction.

Figure 2.2.7. AJJ control signal over the first period: roll, while the joystick is fully pushed in the left direction.

+ negative length idle value is 2ms (Fig 2.2.4, c). Slowly pushing the throttle joystick up will linearly decrease the throttle parameter period length from its 2ms idle value to 1ms maximum value, corresponding to the maximum amount of throttle you can give to the MAV.

Signal summary:
- Measured pulse width value: \( u = 280\mu s \)
- Measured ground-to-control Tension: \( U_e = 2.89V \)
- Measured ground-to-power supply tension: \( V_{cc} = 2.76V \)
- Measured frequency: \( f \approx 240Hz \)
- Measured rising edge delay: \( d_u \approx 2\mu s \)
- Measured falling edge delay: \( d_f \approx 4\mu s \)

2.3. Sanhuan Copter MaxZ Swift 6020-1

2.3.1. Description

The Copter MAX-Z Swift, model 6020-1 manufactured by Sanhuan [7], is a 3-channel R/C helicopter with a built-in Gyro for more stability. It is lifted by a coaxial rotor system. The coaxial configuration of the main rotor allows to counter the angular momentum issue common to all helicopters [6]. A small tail rotor is introduced, in this case not to provide a constant input of angular...
acceleration in the opposite rotation direction to that from the main rotor, but to modify the pitch of the body. Hence it is also coaxial to the two main rotors. You can control the helicopter in 3 directions & around 2 axis: The throttle, as well as the pitch and the yaw. In the case there would be some residual Angular momentum left, you can correct it by using a small rotating switch on upper right corner of the controller. The roll command parameter is not provided here. One of the main differences (and advantage) compared to the Alien Jump jet, apart from the fact that it is a coaxial helicopter, is that it is mechanically stabilized by a weighted flybar attached to the main rotor, and therefore given the moment of inertia, it is much more stable than the Alien Jump Jet. The helicopter flies as stable as a floating object, you can easily control it to land on any spot you want, making it potentially very interesting in the scope of this project.

The controller is a 3 channel transmitter, powered by 6 AA batteries. The helicopter can be charged directly via a cable on the transmitter. In addition, the 6020-1 includes a USB charging cable so that the battery can be charged without connecting the helicopter to the transmitter.

2.3.2. Specifications

- Product model number: 6020-1
- Manufacturer: Sanhuan Ltd [7]
- Motors: 2
- Weight: 41g
- Length/Width/Height: 195mm/45mm/95mm
- Main rotor diameter: 173mm
- Tail rotor diameter: 32mm
- Battery: 130mAh LiPo battery
- Flight Time: Around 10 minutes
- Control: Proportional 2-axis 3-channel Infra-red (THROTTLE, YAW, PITCH)
- Stabilization: 2-axis auto-zeroing computer gyroscopes
- Power Input: 100v to 240v AC - 5 watts
- Handset: 3CH IR strong plastic with 2 a yaw factor (angular momentum) corrector switch.
- Included: 1 Helicopter, 1 Handset, 1 Charger, 2 main rotors & 1 tail rotor spare parts, a screwdriver.

2.3.3. Retro-engineering the MaxZ helicopter

2.3.3.1. Preliminary Notions

Like the one of the Alien jump jet, the Maxz Copter handset has three controls operated by moving the two joysticks either up and down or side to side. The left joystick, controlling the throttle, is restrain to move only in the vertical direction because there is not a roll parameter in the MaxZ Copter. The handset provides also a small rotating switch that offers the possibility to correct the residual angular momentum in case there would be a small error in the gyroscopes calibration. The gyro is an essential component in many RC helicopters and are most commonly used to control unwanted movement on the yaw axis or «residual angular momentum» that would have not been corrected by the coaxial main rotor configuration. The yaw gyro’s job is to sense any undesired rotation around the yaw axis (clockwise or counter clockwise rotation around the vertical axis), and to automatically correct the orientation of the helicopter. Without a yaw gyro, even if the helicopter was trimmed out to fly straight initially, it would eventually begin to drift and rotate right or left. This is very important while considering the retro-engineering method with whom we will proceed, as the first strategy described in the 2.1 section, consisting of directly generating the motors impulsions, would appear to be very risky in this case. If the gyro’s are not enough, the designers also add a small rotating switch
2. Reverse Engineering

Figure 2.3.1. Tentative of retro-engineering the handset of the MaxZ Copter

to correct any residual yaw. Rolling the switch up will add a yaw bias in the left direction, while rolling it down will add a yaw bias in the right direction.

The gyros stabilizers have also another purpose: Unlike the Alien Jump jet, the MaxZ Copter will immediately and automatically return to a stable horizontal flying state and stop moving when the handsets joysticks are centered to their idle positions (except of course the throttle). The neutral joystick position mean that the helicopter will "return to initial stable state" which is the horizontal position. Therefore if a rapid stop is needed, the pilot must only let the axis joysticks return to their initial default position.

We will here use the experience acquired by retro-engineering the Alien Jump Jet in order to get the result we want.

2.3.3.2. The handset

Exactly like for the Alien Jump Jet, the control signal of the MaxZ Copter is constructed directly in the Handset’s PCB. It is therefore only possible to apply the strategy 2 or 3.

2.3.3.3. The Flyer

Similarly to the Alien Jump Jet, in the belly of The MaxZ Copter we can clearly see the communication photodiode used to receive the infra-red signal. We can also see the PCB, the various chips, but unlike the Alien jump jet, there is not any serial number we could use in order to get information on the photodiode. will therefore have to get the functioning voltages as well as the encoding pattern\(^3\) ourselves. The photodiode is connected to the embedded microcontroller through three pins, so we can assume they work the same way as for the Alien Jump Jet: One for the ground, one for the power supply, and one for the control signal. We First we have to determine, by taking various measurements between each connections of the photodiode, which pins corresponds to which parameter (Fig 2.3.2). then by connecting our scope directly to the photodiode correct pins, we were are able to get the signal described in the results section.

2.3.3.4. Signal description

Unlike the Alien jump jet, the MaxZ Copter signal is unfortunately coded. That mean that we will have to decode it or find a corresponding pattern in order to generate the correct control signal. The signal is non consistent, in the sense that the handset send the control signal only when the user moves the joysticks from their initial positions, otherwise there is no communication between the handset and the helicopter. From my understanding of this signal, it is composed as follows:

First, as we consider the signal, i believe that the structure of the signal is rhythmic constant clock, rising-edge/falling edge bias . In this scope, the signal needs to be understood in the following way: each constant time (here

\(^3\) A full description of the PC3388 IR Photodiode is included in the Appendix section of this report.
2. Reverse Engineering

Figure 2.3.2. MaxZ Copter Photodiode visible to the left. The pins description is the following: Control(1), Vcc(2), Ground(3)

Figure 2.3.3. MaxZ Copter: illustration of a signal '0000011111' to demonstrate the 0-to-1 transition.

probably 1ms), the reader wait for a rising edge or a falling edge. Between these clock cycle, the signal immediately sets itself for the next transition, hence the small gaps or pikes (Fig 2.3.3). Bits value can therefore be read by looking the direction of the edge before all small gaps. We consider the '0' to be a falling edge, while the '1' would be a rising edge. The transition between a 1 and a 0 is a simple rising edge without a gap, since the in next clock cycle we should read a falling edge. While the transition between a 0 and a 1 seems to be more complicated (Fig 2.3.3 the middle transition). The overall signal length is 38 milliseconds. And has 32 bits according to this understanding.

Let's go deeper in the signal structure. The signal is separated amongst the various parameters as follows (Fig 2.3.4): First there is an initialization pattern (Fig 2.3.4: initialization) consisting of a 3.75 milliseconds pulse followed by a 1ms negative pulse. After the initialization, we can start reading the rising/falling edges corresponding to the bits values. The first 8 bits (bits 0-7) determine the throttle. On the 2.3.4 illustration, that would give us the value '00000010', as it was taken when the throttle was very low (recall that setting no throttle sends no signal to the Helicopter). Then, when we slowly lift the throttle joystick up, we clearly see a pattern of increasing values, until the maximum: '01111000' (Fig 2.3.5).

Then the next parameters are the yaw (bits 8-11) followed by the pitch (bits 12-15), both of them have values between '0000' and '1111' independently of the direction of the corresponding joystick. As soon as you give yaw/pitch in any direction, these bits values increase. The direction for the yaw is given by another bit: the 16th bit (Fig 2.3.4 :yaw slct). A 16th bit value of 1 meaning «yaw is in the left direction» while a value of 0 meaning «yaw is in the right direction». The direction for the pitch is also given by a single bit: the 17th bit (Fig 2.3.4 : pitch slct). A 17th bit value of 1 mean «pitch is in the down direction» while a value of 0 meaning «pitch is in the up direction». Idle values for both selectors bits are a bit value of '1'.
2. Reverse Engineering

Figure 2.3.4. MaxZ Copter signal description summary. When we set the throttle joystick to a very small value, all other parameters are left to 0.

Then the next 6 bits (bits 18-23) determine the yaw bias, controlled by the small rotating switch in the top right corner of the handset. This is probably a constant that will be added to the yaw parameter in order to construct the final motor yaw signal. Giving a maximum bias to the yaw in the right direction will produce a signal bit value of '111010' (Fig 2.3.6), and will counter any residual helicopter rotation in the left direction. While setting it to the maximum value in the left produce the signal bits value '011100' (Fig 2.3.7), and will counter any residual helicopter rotation in the right direction. I think that the bias value is a signed integer, as when we try to set the bias switch to a median position, we can see the yaw bias first bit change from '0' to '1'. Therefore this part seems to be a signed-magnitude integer representation, however the result we will later see didn’t match to such a representation.

Finally the last 8 bits (bits 24-32) represent the checksum. I think that it is a standard cyclic redundancy check (CRC) checksum. However, the calculations I did didn’t match with this interpretation. An early interpretation of this non-consistent result would be the following: Since I wasn’t able to figure exactly the yaw bias part of the code, it is possible that the limit between the yaw bias & the checksum that I figured isn’t at the right place. Indeed a 5 bits length yaw bias would make more sense than the 6 bits i used, since the yaw parameter is only 4 bits. There is no logic in putting 6 bits for a bias on a 4 bits parameter. Nonetheless the reason why i gave the yaw bias this size come from the results I got for the length of the checksum when moving carefully each parameters and looking which part of the signal was not constant. Indeed we assume that the checksum is computed with all the parameters of the code. When I was moving the throttle, I saw the last 8 bits change, hence my interpretation. I now assume that there has been a mistake while reading the results in the oscilloscope that could have led to a false interpretation, hence the results.

Signal summary:
- Estimated «in code» clock cycle: 1ms
- Measured ground-to-control Tension: $U_c = 3.12V$
- Measured ground-to-power supply tension: $V_{cc} = 3.04V$
- Measured frequency: $f \approx 500Hz$
- Measured rising/falling edge delays: $d_u \approx 4\mu s$
2. Reverse Engineering

Figure 2.3.5. MaxZ Copter maximum throttle signal. When reading the first 8 bits we get the value '01111000'.

Figure 2.3.6. MaxZ Copter maximum throttle signal. When adding a maximum yaw bias in the right direction (rolling the switch down in the handset). The value we read here for the bias is '111010'.

Figure 2.3.7. MaxZ Copter maximum throttle signal. When adding a maximum yaw bias in the left direction (rolling the switch up in the handset). The value we read here for the bias is '011100'.
3. Interface implementation

3.1. Computer interface description

The computer interface has been implemented for Windows x86/x64 platforms. It was successfully tested on a Windows 7 64bits platform.

3.1.1. Cahier-des-charges

The goal of the PC interface is to be able to communicate with the Flyer in real time, while using a gamepad signal as input. Therefore it is developed around the following central points:

1. The program will take as inputs the signal from a USB universal gamepad, in this case the Saitek P880 [8]. More precisely we will take the values of the movement of the two joysticks as the main parameters for Throttle, Roll, Pitch and Yaw. In the case where some of these parameters are obsolete (for example Roll in the case of the MaxZ helicopter), the software still takes the corresponding values and construct the control parameters accordingly. There should be no exception cases treated by the software.\(^1\)

2. The software then construct a corresponding control string containing all the control parameters values depending on the current joystick positions. The string should, in a first time, look like \(T<x>P<y>Y<z>R<w>\) with. \(x, y, z\) and \(w\) corresponding to the current Throttle, Pitch, Yaw and Roll respectively. Once constructed it will send the string in the output channel to the MAV.

3. The interface should come as handy as possible. Since it will be used with different types of MAVs (helicopters, quadcopter, flappers, jets) which all have different controls handling, the software will include the possibility for the user to easily map the different parameters (i.e. throttle, yaw, pitch, roll) to the various joysticks of the gamepad at will, giving him the opportunity to reproduce the original commercial handset joystick configuration of the MAV he will be using the software with.

4. The interface should come as easy to use and understand as possible, further improvement should be easy to make in order for the software to be easily reusable in other projects or studies.

5. The software should be as compatible as possible. Adapting it to another gamepad than the one which was provided should be as easy as possible, as well as adapting it to other communication channels like Bluetooth or WiFi.

3.1.1.1. Inputs

The provided Gamepad, Saitek P880, input signal on a USB port. We will read the value from the windows event stack using the SFML library [9].

3.1.1.2. Outputs

A command string containing all the control parameters bound for the MAV embedded microcontroller. In order for it to be able to handle the command properly, the string should be a standard ASCII string with the following structure:

\[1\] Here we choose to ignore the fact that not all MAVs have the same parameters. We handle this in each MAV microcontroller that we will implement, as the software needs to be as compatible & adaptable as possible.
3. Interface implementation

With the corresponding parameters:

'sC2' This part is specific to the dSPIC Firmware programming and is therefore described in the corresponding section (3.2).
'sgn' The command "signal" indicating that you now send a control string command to the flyer.
'T<x>' an integer between -100 and 100 included, corresponding to the the amount of throttle you want to give to the MAV. An <x> value of 100 corresponds to the maximum throttle motors power of 100%, while a -100 value corresponds to the minimum throttle motors power of 0%.
'P<y>' An integer between -100 and 100 included, corresponding to the amount of pitch you want to give to the MAV. A <y> value of -100 corresponds to the minimum pitch, moving the nose of the MAV downwards, while a value of 100 would correspond to a maximum pitch, lifting the nose of the MAV upwards. A parameter value of 0 would correspond to no pitch, keeping the MAV in its current status.
'Y<z>' An integer between -100 and 100 included, corresponding to the amount of yaw you want to give to the MAV. A <z> value of -100 corresponds to the maximum yaw in the left direction, while a value of 100 would correspond to a maximum yaw in the right direction. A parameter value of 0 would correspond to no yaw, keeping the MAV in its current status.
'R<w>' An integer between -100 and 100 included, corresponding to the amount of roll you want to give to the MAV. A <w> value of -100 corresponds to the maximum roll in the left direction, while a value of 100 would correspond to a maximum roll in the right direction. A parameter value of 0 would correspond to no roll, keeping the MAV current status.

3.1.2. Classes description

The PC interface is called MAVPCI for Micro Aerial Vehicle PC Interface. The version described here is v1.0, the one handed with this report. MAVPCI is organized as follows:

- MAVPCI.cpp: The parent file of the software, containing the main() method, consisting of the sending loop. Each iterations of this loop checks the positions of the joysticks and send the corresponding control signal to the MAV. This loops runs at a speed define by the REFRESHRATE parameter.
- Mapping.cpp: Mapping class, used to create customized associations between the joysticks of the gamepad and the control parameters (i.e. throttle, pitch, yaw and roll).
- HandsetInput.cpp: HandsetInput class, used to describe a Handset/Gamepad. As an independent thread, it accesses the status values of the buttons, and more importantly for us the joysticks in real time. This values are accessible to the user by using the corresponding methods of this class.
- Signal.cpp: Signal class, describes an output signal. Construct the output command string and delivers it to the communication channel.
- COM.cpp: Communication channel. Used to handle the communications through the USB ports. Since the Operating System drivers detects the USB interface as a RS-232 standard (Serial COM) compliant device, we can use a standard Serial COM port to communicate with the USB interface of the microcontroller.

A detailed description of each Module/Class is provided below. The complete source code documentation is handed with this report.
3. Interface implementation

3.1.2.1. MAVPCI.CPP

This is the Front door of the program. This file contains the main() function which perform as follows:

- It first set a mapping (Mapping object) corresponding to the user needs
- It then launches the HandsetInput thread, which will handle the input events (Gamepad)
- It creates a Signal object, which will handle all the output events and communicates with the output channel, in this case the COM module.
- It run the main loop until the "END" key is pressed. The loop runs at a refresh rate defined by the parameter REFRESHRATE (float) corresponding to the time, in seconds, the program waits between each sending iteration. This small delay is crucial in order to avoid an overflow in the receiving side (the dSPIC Firmware) of the COM channel. The default value for the REFRESHRATE is 0.01, corresponding to a sending frequency of 100Hz.

3.1.2.2. MAPPING.CPP

This class handle all the mapping-related functions of the program.

- Remember that the user choose, as stated before, what joystick axis he wants to assign for each parameters in the MAVPCI::main(). Once this has been done, these parameters are saved in an instance of Mapping (in this case called myMap) using the Mapping::SetParameters(std::string throttle, std::string yaw, std::string pitch, std::string roll)

function. This function will control the validity of the inputs and set the corresponding mapping between the parameters & the joysticks of the gamepad. Valid inputs are the following strings:

- 'LX' : left joystick X (horizontal) axis.
- 'LY' : left joystick Y (vertical) axis.
- 'RX' : right joystick X (horizontal) axis.
- 'RY' : right joystick Y (horizontal) axis.

- You can access the current mapping at any time. In order to know what is the current mapping of the Mapping instance you are working with, you will call the Mapping::GetThrottle() Mapping::GetYaw() Mapping::GetPitch() Mapping::GetRoll()

functions. These are useful if you want to know what joystick is currently assigned to a given control parameter.

- If you want to know what are the current values/position of the gamepad/handset joystick corresponding to a given parameter (with respect to the mapping you set), you will call the Mapping::GetThrottleValue(HandsetInput* userHandset) Mapping::GetYawValue(HandsetInput* userHandset) Mapping::GetPitchValue(HandsetInput* userHandset) Mapping::GetRollValue(HandsetInput* userHandset)

functions. For example, if the user wants the throttle to be controlled by the right Joystick, Y axis of the gamepad/handset, he will input "RY" as string in the Mapping::SetParameters() corresponding field. This string value will be saved in the m_Throttle as a class variable, and will be accessible through the Mapping::GetThrottle()
function. Then, at any time, to access the current position value of the joystick corresponding to throttle (with respect to his mapping, which is now RY - Right joystick Y axis for throttle), of the current gamepad/handset he is using, he will call the

Mapping::GetThrottleValue(HandsetInput* userHandset)

function, which will return the current position of the right joystick Y axis of the HandsetInput (Gamepad) he is using, as he previously set it. Note that the returned value are signed integer between -100 and 100.

3.1.2.3. HANDSETINPUT.CPP

This class handle all the inputs (handset/gamepad) related functions of the program. It is organized as an independent thread from the main() method in order to not be slowed down by the REFRESHRATE of the MAVPCI::main() loop. Indeed, here we want to access the current state values of the handset/gamepad joysticks in real time. This class uses the SFML (Simple and Fast Multimedia Library) to have access to the USB Multimedia input event stack [9]. Each HandsetInput instance is created with a corresponding handset ID, which corresponds to to the Multimedia Joystick Port ID in the Operating System. That means that the user can creates multiple handsets/gamepads interfaces simultaneously, each being independent HandsetInput object. The HandsetInput objects works as described here:

- The thread is launched when the HandsetInput::launch() function is called on an HandsetInput instance.
- When launched, the thread creates a sf::Window instance, which in this version is not used to display information, but could still be used to display various controls-related information in further versions. Note that even if it is not used, the sf::Window object is necessary in order to access one of its class variables : the sf::Input instance. Indeed these instances cannot live by themselves, they must be attached to a window. In fact, each sf::Window manages its own sf::Input instance, available as soon as the window is created. Getting a reference to the input associated to a window is done by the function Window::GetInput.
- Once an instance of Window has been created, you can access the event stack by using the Window::GetEvent(Event) trigger. the Event stack concerning the joysticks can be accessed using for example [9]:
  Event.JoyMove.JoystickID
  Event.JoyMove.Axis
  Event.JoyMove.Position

However, this event system is good enough for reacting to events like window closing, using a single key press (in our case we use it for the END key event). But if you want to handle the continuous motion using a joystick movement, then you will soon see that there is a problem : there will be a delay between each movement, the same sampling delay defined by the operating system when you keep on pressing a keyboard key. Hopefully SFML provides easy access to real-time input, with the sf::Input class. That is the reason why we need the sf::window object.
- In our case, we directly access the sf::Input instance of the sf::Window object by using a reference to this class variable, using the method stated in the previous point. We are now able to work with this sf::Input instance to get the results we need.
- We can then select which input source we want to use (Keyboard/Mouse/Joystick) using the corresponding functions, and which key of this source you want to work with using the corresponding IDs. In our case :

  Input::GetJoystickAxis(int HandsetId, int JoystickAxisID)
function. Note that the IDs have been enumerated in the SFML library in order to be easier to use\(^2\). You can therefore access them using the corresponding string macros.

- We save the current values (int) of the two joysticks positions in the class variables \( m_{Rx} \), \( m_{Ry} \), \( m_{Lx} \) and \( m_{Ly} \). The values saved are integer values between \([-100:100]\), \(-100\) being the leftmost/downiest value and \(100\) being the uppermost/rightmost position for a given joystick of the gamepad. Note that in SFML the joysticks are not distinct objects. Indeed in the SFML scope the distinction is made between different axis. If you change gamepads, you will probably need to test all possible \(JoystickAxisID\) to get the ones with the ID corresponding to the gamepad manufacturer’s assignment. There are 8 axis available per handsetID. In our case we have 2 joystick in the handset, which gives us 4 different \(JoystickAxisID\). (Other Axis are all set to an default integer value of 0).

3.1.2.4. SIGNAL.CPP

This class handle all the outputs-related methods. It constructs the corresponding output signal from the control parameters as described in the Cahier-Des-Charges (section 3.1.1.2) and deliver it into the communication channel, which in this case is a standard RS-232 serial COM port output. See the \(COM.cpp\) class description for more details on the communication channel itself. Since each signal is created with its own \(comID\), corresponding in this case to the COM port you will use with this Signal object, you can create multiple signals at the same time by instantiating them with different \(comIDs\). That mean that you can handle multiple flyers with multiple (or the same) HandsetInput/Mapping at the same time. Note that once a Signal object has been created, you cannot change the \(comID\) parameter anymore.

The Signal object works as follows:

- To create an instance of Signal, you need to already have a Mapping object (which means the user already specified the mapping he will use), an HandsetInput object, and a \(comID\), designating (in this case) which COM port you will use with this instance of Signal.

- Once a Signal instance is created, the first thing to do is to open an output communication channel. This is done by using the Signal::openComport() method. This method sets the COM communication channel on the specified \(comID\) port. It then clears the dSPIC receiving bufer by sending the “\n” character. Once you are finished with a COM port channel, don’t forget to close it, using the Signal::closeComport() method.

- You can update the value of the controls parameters to be sent whenever you want and at the frequency you want, by using the Signal::updateSignal() method. This function saves the current HandsetInput object joysticks positions, with respect to the Mapping, to the class variables: \(m_{CurrentThrottle}\), \(m_{CurrentYaw}\), \(m_{CurrentPitch}\) and \(m_{CurrentRoll}\) as integers between \([-100:100]\). For a more user friendly interface, the negatives values for throttle joystick are ignored. Therefore we construct \(m_{CurrentThrottle}\) differently from the other parameters. Check the source code documentation to read more about this.

- Once you saved the control parameters positions you want to send in the class variables, you can send them to the communication channel (Serial COM), by using the Signal::sendSignal() method. this function will construct the string to be sent as described in the Cahier-des-Charges (section 3.1.1.2), convert it to a char table, and send it through the communication channel (Serial COM).

3.1.2.5. COM.CPP

Basic functions for sending and receiving data on port RS232.

\(^2\) A full description of these macros is also available in the SFML official website.[9]
3. Interface implementation

- **BOOL OpenCOM (int nId)**: Open the COM port specified by the parameter nId.
- **BOOL CloseCOM ()**: Close the previously opened COM port.
- **BOOL ReadCOM (void* buffer, int nBytesToRead, int* pBytesRead)**: Receive data through the COM channel.
- **BOOL WriteCOM**: Send data through the COM channel.
- **int COM_test()**: This function is provided to test the COM port communication module behavior. It is not used in the MAVPCI software. However, it is very useful if you want to integrate this COM module into another project, or simply test the behavior of your COM ports while communicating with a device. You can use it by simply calling the COM::test() from anywhere in your code. It will then ask you what port you want to use, and then propose you to send/read data through that COM port.

As this class is the communication channel, and in order to be able to easily switch from this Serial COM to any other communication channel type (Bluetooth or WiFi for example), it is made of global functions only. This is made to allow the user to simply add his communication channel handling files and modify the Signal class accordingly.

Note: This is greatly inspired from various Internet sources and tutorials. An unsuccessful attempt to transform this C module to a full C++ object has been made. However, it didn't seem to work because of some `<window.h>` library incompatibilities with the original C Class structure.

3.1.3. Libraries/Compiler constraints

The MAVPCI 1.0 software uses SMFL library version 1.6 [9].

It is important to denote that the release of SFML library used in this software was compiled with minGW gcc 4.4 compiler, which makes the SFML libraries not compatible with the old minGW gcc 3.4.5 compiler (used as the default compiler in many IDEs, like Code::Blocks, the IDE used for this software). Therefore, you need to compile this software using AT LEAST gcc 4.4 compiler [10] in order to make it work properly.

3.2. Embedded Firmware interface description

3.2.1. Cahier-des-charges

The flyer we want to take control of will have to carry with him our own PCB. This PCB, a USB-BT-RS485 was provided by Rico Möckel from the Biorob lab (Appendix 2). It includes a programmable 16bit microcontroller running at 40Mhz. The programmable firmware is already provided with a USB port handling structure and all the necessary communications protocols (USB). We will therefore have to implement an extension handling the following tasks: Communications between the PC interface and the embedded microcontroller on one hand, and the construction and injection of the corresponding electronic control signal in the flyer on the other hand. Recall that we consider here the strategy 2 described in the 2.1 section of this report, applied to the Alien Jump Jet. Thus, the following describes the firmware extension implemented for the AJJ embedded microcontroller.

Since base functionality like the communication channel where already provided with the chip as long with their documentation, we will only fully describe here the extension implemented in this project. For further description on the parts I didn't implement, refer to Rico Möckel, the project supervisor.

---

3. A C file from which this module is inspired is available on http://www.cppfrance.com/code.aspx?id=22441. This source is itself inspired from various other Internet sources.
3. Interface implementation

The microcontroller extension is developed around the following central points:

1. The microcontroller will take as input a command string form the PC interface coming through the USB communication channel, containing a control string with the value to which the user want to set the parameters (i.e. throttle, yaw, pitch and roll). The microcontroller will ignore the non necessary parameters, for example the roll in the case of the MaxZ Copter.

2. The microcontroller will then construct the corresponding electronic signal and send it trough the PIN28 output pin. This pin will be connected to the commercial flyer original microcontroller, using the Control Tension($U_e$) described in the section 2.3.3.4, in order to provide full control over the flyer. See the Microcontroller description in Appendix 2 to have an complete description of the pins allocation. The signal construction will be done in a Moore Finite State Machine.

3. The embedded interface should come as handy as possible. Since the goal is to be able to adapt it to different types of MAVs (helicopters, quadcopter, flappers, jets) which all have different controls signals handling, the code will include the possibility to easily be updated. Therefore all the signal construction methods (which are specific to the Toy flyer you consider) will be implemented in a separate .C file (in this case AJJ.C) in order to provide the ability to easily switch between different signals constructions by just adapting the corresponding C file.

4. It should come as easy to use and understand as possible, further improvement should be easy to make in order for the software to be easily reusable in other projects or studies.

Note that all time values in the microcontroller are set in a 20 microseconds unit scale. This comes from the fact that we need to have unit time small enough to work without losing precision, but big enough to avoid 16bits integer overflows during the execution of the code. The scale of the clock is set in the cong.h file.

3.2.1.1. Inputs

The MAVPCI PC interface command string, $sC2sgnT<x>P<y>Y<z>R<w>$ as described in the 3.1.1.2 section.

3.2.1.2. Outputs

The electronic control signal inherent to the Alien Jump Jet MAV, as described in the retro-engineering section 2.2.3.

3.2.2. Class description

The essential classes to understand the extension are described below:

- **main.c**: The microcontroller runs a constant loop in the main() method, after initializing it’s working parameters. We create a timer variable timer_sendtime during the initialization of the microcontroller. This variable will contain the time of the next event. Indeed it will be the variable against which we test the current time in order to determine if we have to move to the next state in the FSM. Once the microcontroller runs the main loop, the timer_sendtime pointer will always be updated depending of the current state of the finite state machine generating the output signal. During the initialization, the microcontroller also creates an instance of $Flyer$ object, called AJJ, as described in the AJJ.h file. This object contains the various control & state parameters. On the main loop, the microcontroller is doing two things: First it search for an input command on the USB communication port, by executing each iteration the $sacp::sacp_main()$ method. This will update the received values in the ringbuffer available for us to use for the output construction. Then it executes the signal constructing method: $ajj::ajj_main()$, which is a finite state machine constructed around the AJJ object described below.
3. Interface implementation

- sacp.c: The receiving module of the controller. Each time a signal is received through the USB channel this module stores it in a ring buffer (ringbuffer.c) and launches the command table module command.c which will verify the contents. The command needs to start with '<k>C2' in order to be recognized by the controller. This corresponds to a query/set type transmission '<k>': «What type of query is this?» Here, an 's' parameter value of means «set», while an '?' parameter value means «query». The standard value of this field for our sg command will be 's'.

C2: That is the controller ID to which you address your command. Here, our controller has the ID 2 therefore we will always set this to 'C2' when communicating with our controller.

- command.c: This is the command table of the receiving module, sacp.c. Each time the controller receive a command it will execute the method corresponding to the command as described in the following table. Each command string need to start with '<k>C2' as described in the previous point. These are the main useful commands for us:

- 'sgn': This command executes the function command_signalreceive(). It handles the packet containing the control parameters (yaw, pitch, roll, throttle). This function will extract the parameters from the received packet in the buffer, using the ASCII to integer converter ascii::a2i() function. Once extracted, the parameters are still not usable (recall that they are integers between [-100-100]). Therefore we need to convert them to the corresponding time values that the Finite State Machine will use to construct the electronic output signal. This will be done by using the ajj::stdconv_func() method.

- 'help': This command executes the function command_help(), which print all commands in the USB communication channel

- 'led<k>': This command executes the function command_led(), which deal with the LED control. '<k>': '1': enable, '0': disable, 't': toggle

- 'test': This command executes the function command_test(), test function which just return the standard answer: '*'.

- 'version': This command executes the function command_firmware(), print the firmware version in the USB communication channel.

- 'int<k>/float<k>': These were test functions implemented to test the conversion of integer/floats to ASCII and return the results through the USB communication channel. '<k>': int/float you want to convert.

- ajj.c: This module contains all the output signal construction related methods. The electronic output signal is generated in a Moore Finite State Machine, with each state corresponding to a step in the sequential construction of the signal (Fig. 3.2.1). Variables STATE & NEXSTATE of the global Flyer object instance (described by an enum in the ajj.h), describes the current & the future state of the FSM corresponding to the ajj_main(unsigned int* timer_sendtime) function.

The FSM itself works as follow: On each iteration of the main() loop, we look on which state we are. Then we check the timer_sendtime in order to know if we have reach an event time. Event here means that we have to change the current output (from '0' to '1' or vice versa), thereby moving to another state. Since the state transition is deterministic through time, on each event we simply set the nextstate with respect to the current state (i.e. if the current state is 'yaw0' & we have reach the event time, the next state will be 'Idle1'. If the current state is 'roll1' and we reach the event time, the next state will always be 'roll0'... See Fig 3.2.1 to have a summary of the state transitions). We also set the timer_sendtime to contain the time to the next event. This means that we set how long the FSM will stay in the nextstate while still being in the current state. We set this time (timer_sendtime) with respect to the current control parameters input values, stored in the Flyer object variables m_Throttle, m_Pitch, m_Yaw & m_Roll of the Flyer object instance. At the end of each output signal cycle (When the five parameters have been sent) when we reach the 'Idle1'
3. Interface implementation

Figure 3.2.1. Summary of the AJJ.c Finite Machine states transition. States are in red. Each state corresponds to an output of 0 or 1. The blue dotted lines are the «event» times in the FSM.

In state, we set the time for which we are going to stay in 'idle0'. This time corresponds to the time to the next cycle, thus corresponding to the output frequency. This output frequency is set in the REFRESHRATE constant. This constant is set on the Flyer instance creation to be 20ms, corresponding to 50Hz. At the end of each FSM iteration we set the microcontroller output PIN28 to the output value ('0' or '1') corresponding to the current state.

The ajj::stdconv_funct() converts the parameter values from integer values between [-100:100] to time values oscillating between 720 and 1720 microseconds (Recall each control parameter period in section 2.2.3.4) and store them in the Flyer object global variables m_Throttle, m_Pitch, m_Yaw & m_Roll. Since the unit time is 20 microseconds we have to convert these values from integers between [-100 100] to integers between [36:86]. Indeed, $36 \times 20\mu s = 720\mu s$, $86 \times 20\mu s = 1720\mu s$. This is done by the following linear equations (recall Fig. 2.2.4 description):

$$t_{Throttle} = 36 + \left\lfloor \frac{200 - (100 + Throttle)}{4} \right\rfloor$$

$$t_{Pitch} = 36 + \left\lfloor \frac{200 - (100 + Pitch)}{4} \right\rfloor$$

$$t_{Roll} = 36 + \left\lfloor \frac{100 + Roll}{4} \right\rfloor$$

$$t_{Yaw} = 36 + \left\lfloor \frac{100 + Yaw}{4} \right\rfloor$$

- $t_{Throttle}$: Electronic output signal : The width of the time interval corresponding to the throttle parameter.
- $t_{Pitch}$ : Electronic output signal : The width of the time interval corresponding to the pitch parameter.
- $t_{Roll}$ : Electronic output signal : The width of the time interval corresponding to the roll parameter.
- $t_{Yaw}$ : Electronic output signal : The width of the time interval corresponding to the yaw parameter.
- $Throttle/Pitch/Roll/Yaw$ : Input values, as received from the command string input: The joysticks position values, integers between [-100:100].

These equations induce a discretization factor of 2. Indeed, while we had 200 possible samples [-100:100] on the PC interface end, we now construct the electronic output signal within a sample of 50 different values [36:86], because of the time scale (20 microseconds) in the microcontroller.
4. Final results

4.1. Alien Jump Jet

The results we obtained with the Alien Jump Jet are quite good. Indeed, using the MAVPCI software along with the USB-BT-RS485 Firmware implemented according to the description above, we were able to make the quadcopter take off and respond to our commands (Fig 4.1.1). The interface overall works as expected, with the exception of a small delay bug introduced periodically by the microcontroller. It seems that each five to ten seconds the microcontroller experience some lag end misses the event trigger, resulting in an incomplete electronic output signal construction. This was confirmed by oscilloscope measurements.

One reason could be that the PC interface sending rate through the USB communication channel is too high (default setting of 100Hz, section 2.1.2.1), triggering an overflow in the microcontroller’s end. However, we see that setting the sending rate to a 10Hz value in the MAVPCI software do reduce the problem, but does not solve it. Indeed, when we measure the signal value in the output PIN28 with an oscilloscope while the microcontroller doesn’t receive any packets on the USB channel (therefore measuring the standard idle signal), the microcontroller does not appear to trigger the error. That lead me to think that this error must be related to the communication channel handling in the Firmware implementation, since the error only appear when there is traffic between the PC interface & the embedded microcontroller. Therefore I think that this bug is induced by the combination of an overflow problem somewhere in the chip, triggered by the reception of packets through the COM link.

4.2. MaxZ Helicopter

The MaxZ retro-engineering results are, although incomplete, quite promising. We were able to get almost all the code construction pattern, with the notable exception of the checksum. Breaking the checksum code is, i think, only a matter of time. Unfortunately i wasn’t able to finish that part within the time frame allotted for this project. Once the last part of the signal will be decoded, we should to be able to adapt the current AJJ Firmware Finite State Machine, in order to generate the MaxZ Copter control signal instead. I estimate the necessary time to do these different tasks to be between 2 to 3 weeks.
Figure 4.1.1. Final result: Alien jump jet connected to the PC platform through the embedded microchip.
5. Conclusion

5.1. Discussion

Looking back on the aims of the project, we see that the main goals have been more or less achieved. Indeed, with the insight given by this project, we can now see that it is possible to develop our own robotic control platform, based on common commercial toy flyers platforms available at a relatively cheap price. The full implementation of a control platform based on the Alien Jump Jet demonstrate this. Even if in some cases like for the MaxZ Copter, we hurt ourselves against coded control signals, I still believe that given enough time and knowledge, such difficulties can always be overcome. I also think that in a future perspective our best chance to hack & control such platform resides in the fact that the goals pursued by the toy flyer developers are aimed for commercial use. Therefore they are always in the race to the cheapest price, which is synonym for us to improved simplicity. The simpler the commercial flyers get, the more useful & precious they become in the scope of developing our MAV studies platform. I therefore think that the experimental research towards the way of Hacking and Controlling toy flyers should be pursued, mostly in order to provide cheaper alternatives for schooling & research in the stammering field of flying robots.

5.2. Future work

As a future work it should be very interesting to start testing the Alien Jump Jet configuration with various stabilization algorithms, as controlling it with the original pad as well as with this PC interface proved to require some skillful hands. Such development would be the first steps toward a fully automated, commercial based toy flyer. Another interesting development would be to extend the work already done to other toy flyers models. This would widely increase the scope of this project by offering various alternatives for the potential applications.

5.3. Acknowledgments

When I started working on this project I had no insight on how to perform retro-engineering, as I never really had the chance to apply what I have been learning in the almost only theoretical courses I have been following. Facing day-to-day problems in electronics, informatics and sometimes aeronautics was very interesting and exciting, and now prove to have been a richly rewarding experience, as I now have an insight on how the development process occurs in a laboratory environment, starting from a simple concept, ending with a working prototype. I therefore want to thank all the people in the BIOROB lab who would always be at hand with good and reliable practical and theoretical advice. I also particularly want to thank my project’s supervisor, Rico Möckel, for backing and assisting me in this semester project. I learned a lot of practical things that will undoubtedly help me in the future, starting with the understanding of \LaTeX{} environment necessary to write this report, and through various things like programming in C++ as well as in C, or using an oscilloscope to read and understand electronic signals, and I am therefore very grateful to
him for his support. I also want to thank the professor Auke Ijspeert, head of the BIOROB, for allowing bachelor students like me to perform studies in a laboratory environment. As I finish writing this lines, I really hope that the contribution given by this project will be useful for future studies, and that I will be able to continue working in the robotic field, as I really want to know how further improvements of my project would look like.
References

WEBSITES

[1] Wikipedia, Internet encyclopedia project:
   - Pulse width modulation signal description: http://en.wikipedia.org/wiki/Pulse-width_modulation
   - Cyclic redundancy check (CRC) description: http://en.wikipedia.org/wiki/Cyclic_redundancy_check
[2] EPFL Laboratory of intelligent systems (LIS)
[3] MAVSTAR
   - Micro aerial vehicles for search, tracking and reconnaissance: http://www.robotics.unsw.edu.au/mavstar/
[4] Delft University of Technology:
   - Delfly project: http://www.delfly.nl/?site=DIII&menu=info&lang=en
   - Official website: http://www.snellflight.co.uk/jumpjet.htm
[6] Wikipedia, Internet encyclopedia project:
[7] Sanhuan Swift Max2 Copter
   - Official website: http://www.sanhuanstop.com/
   - Descriptive website on toy flyers: http://www.hobby-ving.com/
[8] Saitek P880 controller pad
[9] SFML Library
   - Full description of the library: http://www.sfml-dev.org/tutorials/1.6/index-fr.php
[10] MinGW open source compiler
    - Official website: http://www.mingw.org/
[11] ETH Zurich, Autonomous system labs
    - Official website: http://www.asl.ethz.ch
    - Reely website: http://www.reely.ethz.ch/index.html
[12] Information for the described Flyers gathered from:
    - ParkZone Vapor official website: http://www.parkzone.com/Products/Default.aspx?ProdID=9223380
    - WooWee Flytech Dragonfly official website: http://www.woowee.com

BOOKS/PUBLICATIONS

Will Regan, Floris van Breugel, Hod Lipson 2006 Towards Evolvable Hovering Flight on a Physical Ornithopter Computational Synthesis Lab Cornell University, Ithaca NY 14853, USA
http://ccsl.mae.cornell.edu/papers/Alife06_Regan.pdf


Innotech Systems inc A Primer on Remote Control Technology, 320 Main Street Port Jefferson NY 11777

I. O'Connor, F. Mieville Machines à états finis (FSM : Finite State Machines), Dépt. EEA, Ecole Centrale de Lyon
Appendix

Appendix 1: MAVPCI user guide
1 Software description

This software has been implemented for a student project called "Hacking & controlling toy flyers" in the BIOROB lab. The goal of this software is to be able to control various toy flyers on which a retro-engineering process have been previously performed, by sending control strings to their embedded microcontrollers. In order to construct the string we use a universal USB gamepad, the Saitek P880 controller (http://saitek.com). And an Alien Jump Jet toy flyer with an embedded dSPIC microcontroller programmed to handle the string control packets.

The software generates control strings of the type

"sC2sgnT<x>P<y>Y<z>R<w>"

<x>,<y>,<z>,<w> : integers between [-100:100] read the description on the Final Report Section 3.1.1.2 for further description on the string structure

and send them through the specified COM port which in this case is an USB port with the toy flyer’s embedded firmware connected to it. The software has been successfully tested with an EFlight’s Alien Jump Jet toy flyer enhanced with an embedded microcontroller to handle the string control packets.

A full HTML documentation is provided with this software.

2 System requirements

To run this software properly on your computer you need

- Windows 7/XP (tested on x64 versions)
- Saitek P880 USB controller with the up-to-date drivers (get them from http://saitek.com/uk/down/drivers.php), or an equivalent.
- a dSPIC microcontroller to which you will send the control commands.
3 Installation


2. Plug the dSPIC interface on a USB port and let the Windows automatic System Device configuration tool install the default Microsoft drivers.

3. Run the software executable MAVPCL.exe

4. Enjoy.

4 Software operation.

1. Set interface environment parameters:
   Once loaded the software will prompt the user to set the various interface parameters:

   (a) Enter the main loop REFRESHRATE: This is the time, in seconds, the program will wait until it sends the next string control packet to the flyer’s microcontroller. You need to enter a float value. The platform was stable with this parameter set to 0.01 second. Do not set the REFRESHRATE parameter too high, as it will lead to overflow errors on the microcontroller’s end.

   (b) Enter the com port ID on which the output packets must be sent. In order to know the port ID the flyer uses when connected to the PC, check the parameters in the "System properties/Material" section of the windows configuration panel.

   (c) Finally the user must set a valid Mapping between the joysticks of the gamepad and the parameters. The prompt will ask the user to enter joysticks + directions for each parameter. Valid inputs are:
      -'LX': Left joystick of the gamepad, X (horizontal) axis. -'LY': Left joystick of the gamepad, y (vertical) axis. -'RX': Right joystick of the gamepad, X (horizontal) axis. -'RY': Right joystick of the gamepad, y (vertical axis) axis. Recommended configuration is: THROTTLE: LY, PITCH: RY, ROLL: RX, YAW: LX, as these are default and more intuitive ones for the Alien Jump Jet.

2. Main sending loop execution
   Once the interface parameters have been set the sending loop will start running at the REFRESHRATE the user entered previously. Just move the Joysticks corresponding to the parameters and the flyer will follow the joystick moves. The command prompt let you see the actual value of the parameters as well as the size of the packet sent.

Once you want to quit the program hit the "END" key and the program will automatically close.
5 troubleshooting:

MAVPCI 1.0 has been successfully tested with an EFlight’s Alien Jump Jet toy flyer enhanced with an embedded microcontroller to handle the string control packets.

If it doesn’t work properly check that the COM port ID is correct as you could be sending packets to another device than the actual flyer you want to send them to.

The software doesn’t work on your machine, refer to the code documentation in order to see how it has been implemented.

The full source code is provided with the software so feel free to adapt it to your needs.

If you have trouble using the software, contact me at bernard.gutermann@epfl.ch

6 Credits

MAVPCI 1.0
software & doc bernard.gutermann@epfl.ch
© BIOROB- EPFL 07.10.2010

The MAVPCI 1.0 software uses SMFL library version 1.6.
© http://wwwSFML-dev.org/index-fr.php

Implemented with CodeBlocks 10.05
© http://codeblocks.org ARR.

compiled with MinGW v4.4 ©http://www.mingw.org/

Special thanks to 'Lynix' tutorial on how to run MinGW 4.4 on Windows
©http://lynix.digitalpulsesoftware.com/2010/06/gcc-4-4-0-sous-windows-avec-mingw/
Appendix 2: USB-BT-RS485 Converter description
Input voltage: 2.3V - 30V
Dropout voltage: 350mV
Output current: 500mA
Stable with 3.3uF output cap

3.3V power supply

MIC5239-3.3YMM

Rico Möckel
dsPIC development board
École Polytechnique Fédéral de Lausanne
Biorobotics Laboratory 01.11.2010

PIC4101
PIC4102
COC41
PIC4201
PIC4202
COC42
PIU401
PIU402
PIU403 PIU404
PIU405
PIU406
PIU407
PIU408
COU4
PIC4201
PIU404
PIU405
PIU406
PIU407
PIU408
PIC4102
PIU401
PIU403
POVin
PIU402
POVIN
## Part List

**Source Data From:** dsPICdevelopment.PrjPCB  
**Project:** dsPICdevelopment.PrjPCB  
**Variant:** None  
**Report Date:** 23.11.2010 11:32:03  
**Print Date:** 23-Nov-10 11:32:10 AM

<table>
<thead>
<tr>
<th>Designator</th>
<th>Comment</th>
<th>LibRef</th>
<th>Footprint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1, C42</td>
<td></td>
<td>22uF, 6.3V</td>
<td>Cap1206</td>
<td>Pol2 1608-10 Polarized Capacitor</td>
</tr>
<tr>
<td>C11</td>
<td></td>
<td>10uF Cap</td>
<td>CAPC1608N</td>
<td>Capacitor</td>
</tr>
<tr>
<td>C12, C13, C21, C22</td>
<td>100nF Cap</td>
<td>CAPC1005N</td>
<td>Capacitor</td>
<td></td>
</tr>
<tr>
<td>C41</td>
<td></td>
<td>1uF, 35V Cap</td>
<td>CAPC1608N</td>
<td>Capacitor</td>
</tr>
<tr>
<td>D11, D12</td>
<td></td>
<td>LED Yellow</td>
<td>RESC1005N</td>
<td>LED</td>
</tr>
<tr>
<td>D21</td>
<td></td>
<td>LED Red</td>
<td>RESC1005N</td>
<td>LED</td>
</tr>
<tr>
<td>D22</td>
<td></td>
<td>LED Blue</td>
<td>RESC1005N</td>
<td>LED</td>
</tr>
<tr>
<td>F1</td>
<td></td>
<td>Ferrit 0.35R 500mA</td>
<td>RESC1608N_R</td>
<td>Ferrite Bead</td>
</tr>
<tr>
<td>Q11, Q12</td>
<td></td>
<td>2SK3547G0L nFET</td>
<td>SSS-Mini-3P</td>
<td>N-Channel MOSFET 100mA, 50V, 12R</td>
</tr>
<tr>
<td>R11, R12, R13, R14</td>
<td>270R Res1</td>
<td>RESC1005N</td>
<td>Resistor</td>
<td></td>
</tr>
<tr>
<td>R15, R16, R17, R18</td>
<td>10K Res1</td>
<td>RESC1005N</td>
<td>Resistor</td>
<td></td>
</tr>
<tr>
<td>R19, R20</td>
<td></td>
<td>100K Res1</td>
<td>RESC1005N</td>
<td>Resistor</td>
</tr>
<tr>
<td>S11</td>
<td></td>
<td>Push button EVPAA</td>
<td>EVPAA Push button</td>
<td></td>
</tr>
<tr>
<td>S12</td>
<td></td>
<td>switch</td>
<td>CVS-02B Double Switch</td>
<td></td>
</tr>
<tr>
<td>U1</td>
<td></td>
<td>dsPIC33FJ128MC802-I/MM</td>
<td>QFNS-MM28_M_SmallExp</td>
<td>High-Performance 16-bit Digital Signal Controller, 128 KB Flash, 16 KB RAM, 28-Pin QFN, Industrial Temperature</td>
</tr>
<tr>
<td>U2</td>
<td></td>
<td>FT232RQ</td>
<td>QFN-32_smallPad</td>
<td>FT232R USB UART</td>
</tr>
<tr>
<td>U4</td>
<td></td>
<td>MIC5239-3.3YMM</td>
<td>MSOP-8 LDO, 500mA, Vin:2.3V-30V</td>
<td></td>
</tr>
<tr>
<td>X11</td>
<td></td>
<td>68016-106HLF Connector 6x1</td>
<td>Connector 2.54x1</td>
<td>Connector 6x1 pins</td>
</tr>
<tr>
<td>X12, X13</td>
<td></td>
<td>TMA-105-01-G-D-5M</td>
<td>Connector Male 2x5</td>
<td>TMA-105-01-G-D-5M</td>
</tr>
<tr>
<td>X14</td>
<td></td>
<td>USB micro</td>
<td>USB micro</td>
<td>Micro USB Connector</td>
</tr>
</tbody>
</table>
Appendix 3: AJJ Photodiode description
IR Receiver Modules for Remote Control Systems

Description

The PC3388 is remote control receiver modules. Pin diode and receiver IC are assembled on one module. Small-sized, light-weight, and low current consumption. modules have been achieved by using resin mold. The demodulated output signal can directly be decoded by a microprocessor. The main benefit is the reliable function even in disturbed ambient and the protection against uncontrolled output pulses.

Features

◆ Supply Voltage Range: 2.7V to 5.5 V
◆ TTL and CMOS compatibility
◆ Photo detector and preamplifier in one package.
◆ Internal filter for PCM frequency
◆ Output active low
◆ Enhanced Immunity against all kinds of disturbance light
◆ No occurrence of disturbance pulses at output pin with in nominal conditions.
◆ Short settling time after power On.
◆ Meet RoHS

Applications

◆ Audio video applications
◆ Home appliances
◆ Toy applications
◆ Remote control equipment

Application Circuit

R-C filter recommended to suppress power supply disturbances. R-C filter should be connected closely between Vcc pin and GND pin.

Block Diagram

Dimensions

1. Output
2. Ground
3. Vcc

Unit:mm
## Recommended Operating Conditions

(Ta = 25°C)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Unit</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating Voltage</td>
<td>Vcc</td>
<td>2.7</td>
<td>5.0</td>
<td>5.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>Input Frequency</td>
<td>fin</td>
<td>30</td>
<td>37.9</td>
<td>60</td>
<td>kHz</td>
<td></td>
</tr>
<tr>
<td>Operating Temperature</td>
<td>Topr</td>
<td>-20</td>
<td>25</td>
<td>80</td>
<td>°C</td>
<td></td>
</tr>
</tbody>
</table>

## Absolute Maximum Ratings

(Ta = 25°C)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply Voltage</td>
<td>Vcc</td>
<td>0</td>
<td>6.0</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>Output Voltage</td>
<td>Vout</td>
<td>0</td>
<td>6</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>Output Current</td>
<td>Iout</td>
<td>0</td>
<td>2.5</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>Operating Temperature</td>
<td>Topr</td>
<td>-20</td>
<td>80</td>
<td>°C</td>
<td></td>
</tr>
<tr>
<td>Storage Temperature</td>
<td>Tst</td>
<td>-40</td>
<td>125</td>
<td>°C</td>
<td></td>
</tr>
</tbody>
</table>

## Electro-optical Characteristics

Specifications hold over the Recommended Operating Conditions, unless otherwise noted herein. All values are at 25°C and Vcc=3.0V/5.0V

(Ta = 25°C)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Unit</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply Current</td>
<td>Icc</td>
<td>--</td>
<td>0.9</td>
<td>1.5</td>
<td>mA</td>
<td>Lin=0 [] , Vcc=3V</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
<td>1.5</td>
<td>mA</td>
<td>Lin=0 [] , Vcc=5V</td>
</tr>
<tr>
<td>Max. Voltage gain</td>
<td>Av</td>
<td>75</td>
<td>80</td>
<td>85</td>
<td>dB</td>
<td>fin=37.9kHz, Vin=30[] p-p</td>
</tr>
<tr>
<td>BPF Bandwidth</td>
<td>fbW</td>
<td>3.5</td>
<td>6.0</td>
<td>8.5</td>
<td>kHz</td>
<td></td>
</tr>
<tr>
<td>Output pulse width</td>
<td>tpW1</td>
<td>500</td>
<td>-</td>
<td>800</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>tpW2</td>
<td>500</td>
<td>-</td>
<td>800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low level output voltage</td>
<td>VOL</td>
<td>-</td>
<td>0.2</td>
<td>0.4</td>
<td>V</td>
<td>Isink=2.0mA</td>
</tr>
<tr>
<td>High level output voltage</td>
<td>VOH</td>
<td>2.7</td>
<td>3.0</td>
<td>-</td>
<td>V</td>
<td>Vcc=3V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.7</td>
<td>5.0</td>
<td>-</td>
<td>V</td>
<td>Vcc=5V</td>
</tr>
</tbody>
</table>
[ Fig. 1 ] Data Signal diagram

Carrier Frequency: 37.9kHz

(Input burst waveform)

(Burst length)

(The minimum Burst length)

(Integrator output)

(Output pulse)

[ Fig. 2 ] Transmitter

S/Transmitter

PD

Vcc

10kΩ

Vout

GND

10μF

10kΩ

30cm

[ Fig. 3 ] Test condition of arrival distance

Transmitter

Vout

OSC

GND

Vcc

Arrival Distance: L

∅: Indicates horizontal and vertical directions

[ Measurement condition for arrival distance ]

Ambient light source: Detecting surface illumination shall be irradiate 200±50Lux under ordinary white fluorescence lamp without high frequency lighting
Electrical/Optical Characteristics

[ Fig.4 ] Supply Current vs. Voltage

![Graph showing Supply Current vs. Voltage](image)

[ Fig.5 ] Sensitivity vs. Supply Voltage

![Graph showing Sensitivity vs. Supply Voltage](image)

[ Fig.6 ] Output Pulse Width vs. Distance

![Graph showing Output Pulse Width vs. Distance](image)

[ Fig.7 ] Directivity (Horizontal)

![Graph showing Directivity (Horizontal)](image)

[ Fig.8 ] BPF Fc Curve

![Graph showing BPF Fc Curve](image)

ESD Test Results

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Conditions</th>
<th>Specification</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Model</td>
<td>C=200pf, R=0Ω</td>
<td>Min ±200V</td>
<td>≥±200V</td>
</tr>
<tr>
<td>Human Body Model</td>
<td>C=100pf, R=1.5kΩ</td>
<td>Min ±2000V</td>
<td>≥±2000V</td>
</tr>
<tr>
<td>Charged Device Model</td>
<td>R=100Ω, 1Ω</td>
<td>Min ±800V</td>
<td>≥±800V</td>
</tr>
</tbody>
</table>

PengCheng Elec-tech Co., Ltd
PC3388