MODELLING THE SAFE CONTROL OF AN AIRCRAFT IN TWO-DIMENSIONS

Nikita Rupani
Carnegie Mellon University
Pittsburgh, PA 15213
nrupani@andrew.cmu.edu

Brian Wei
Carnegie Mellon University
Pittsburgh, PA 15213
bwei1@andrew.cmu.edu

December 14, 2020

ABSTRACT

An aircraft is a complex system composed of a combination of controls from levers and buttons that control its forward motion, pitch, yaw and roll. Our model will focus on the 2D version of aircraft flight control, where the plane can only move in the horizontal and vertical directions. In this project, we model the 2D take-off, flight and landing of an aircraft. By doing so, we aim to understand the complexities and challenges of the formal verification and validation of aircraft systems. Through the process, we navigated several challenges and we were able to formally prove the take-off and landing of our system using the KeYMaera X Theorem Prover , and formulate ideas on how the in-flight control would work. Though we were unable to formally create our in-flight control, we found this project thoroughly exciting and challenging in its course and learned a lot about aerodynamics, modelling physical controllers and the vast field of verification and validation.

1 Introduction

In 2001, NASA proposed the Small Airport Transportation System (SATS) [1] that would revolutionize commercial transportation in the United States. The key idea was to introduce a fleet of small jets that could operate without the need for air traffic control. This idea was heavily funded, but after several concerns surfaced, NASA stopped its research on SATS in 2006. There were many challenges that prevented SATS from becoming a success, from high commercial costs to negative environmental impact to most arguably the most critical aspect – safety [2]. In this project, we seek to understand the critical safety aspects of controlling a small aircraft, similar to one that would have been used in SATS.

But what are the most critical safety aspects to consider when controlling a plane? According to Boeing's statistical summary [4], nearly half of all commercial airplane accidents happen during takeoff and landing, making it the most dangerous part of a flight. In this project we seek to model exactly this – the takeoff and landing of the aircraft To do so, our objective is to create a controller to safely regulate the movement of the plane from point-to-point, abstracting its movement to only the horizontal and vertical directions. We will use the KeYmaera X theorem prover to prove the safety of our model. If successful, this can serve as a basis for more robust models that better model the multitude of real-world conditions to lead to better airplane safety and possibly one day, a SATS II.

We chose to focus on the 2D system because we believe that the challenges of safety are especially highlighted in the horizontal and vertical movement of an aircraft, which are arguably the most necessary directions of motion for an aircraft. Although a 3D system would better capture the full control of a modern aircraft, due to the time constraints of our class project, we will stick to proving safety properties of the 2D system.

1.1 Background and Related Work

During our research stage, we found that modeling flight is a very popular topic in the field of verification and validation. A lot of work has been done on the topic, yet we are still far from provable safe autonomous flight. We looked at several different related works to get a better understanding of the challenges related to modelling flight, from the kind of decisions that go into control to deciding what kind of assumptions are reasonable and the challenges related to aerodynamics when dealing with modelling flight.

One of the projects we looked at was ‘The Engineering Analysis and Design of the Aircraft Dynamics Model For the FAA Target Generation Facility’ [7]. This project presented an extremely thorough model for flight for the FAA (Federal Aviation Administration). The document laid out a very detailed theoretical basis for the model, although did use some simplifying assumptions. For example, we observed that the project reduced the 6 degrees of freedom of the plane to 4 degrees of freedom. Similarly, we decided to capture what we saw as the most important 2 degrees of freedom (forward/backward and up/down) of an aircraft.

Another very interesting paper we looked into was ‘Formal verification of quadcopter flight envelop using theorem prover’ [8]. This paper looked at how factors such as wind and turbulence, altered drone mass and misalignment of thrust output due to damage affected the flight of a quadcopter. We really enjoyed the real-life applications of this research- drones do frequently get damaged and this affects their flight. Similarly, a plane being damaged would affect its flight. However, we decided to stick with the general use case (everyday flight) instead of such a case. At the same time, it was very cool to look at the vast array of applications of formal verification and validation as presented in this paper. It was important to understand that even if we modelled perfect flight, our controller would not be robust to scenarios such as damage or crash.

Another resource we found extremely relevant to our project was ‘Formal Methods and the Certification of Critical Systems’ [9]. Although we did not gain material from this text on how to build our model, it reminded us of the limitations of formal methods for proving the safety of physical flight. This text explains the benefits of formal verification and validation as a way to provide a solid, in-expensive testing ground (particularly useful for flight), but sternly warns of the limitations of computer-software verification vs. real-life (that KeYmaera X theorem prover isn’t robust to the nuances of the physical world). It was worth reading and understanding these limitations to truly understand how challenging it really is to model flight or create a safe autonomous plane in the real world.

Some other resources we looked into were ‘Formal Verification of Flight Critical Software’ [10], which discussed the formal verification tools used to verify the FCS 5000 (Flight Control System), ‘Formal Verification of Curved Flight Collision Avoidance Maneuvers: A Case Study’ [11], which used compositional verification to verify safety properties for collision avoidance in flight and NASA’s ‘Modeling Flight: The Role of Dynamically Scaled Free-Flight Models in Support of NASA’s Aerospace Programs’ [12], which gave us deep insight and background into the extensive world of aerodynamics we needed to learn and consider while creating our model.

2 Model Design

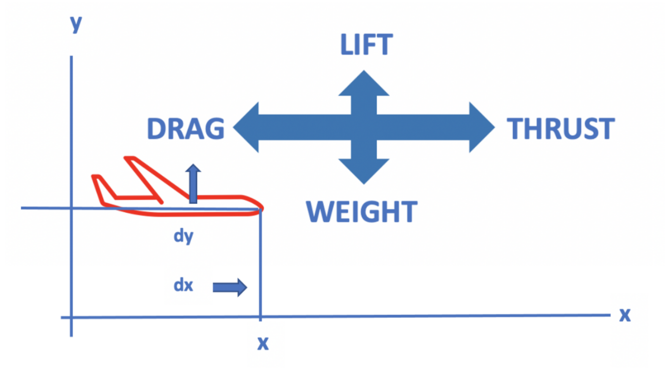
We wanted our model to be precise enough to show the challenges that a pilot faces in controlling the plane’s thrust and elevation when taking off, flying and landing. We are abstracting away ground friction and air resistance since these are relatively minor factors when considering the overall force acting on a plane, given the generally high speeds it travels with.

Simplifying our model just enough to fit our needs, we gave our pilot direct control over the thrust (the forward force generated by the engine), the lift coefficient and the drag coefficient. In reality, the lift coefficient is influenced by a combination of the flaps, slats, and angle-of-attack manipulated by the elevators at the back of the plane which the pilot controls. We felt that giving the pilot direct control over the lift coefficient was a reasonable abstraction to make because the activation of flaps, slats, etc. directly influences the lift coefficient. Similarly, the activation of other special flaps, etc. directly affects the drag coefficient, so we chose to give our pilot direct control over the drag coefficient as well. We are not allowing our plane to travel backwards nor to have negative L, leaving gravity to do the work when we need to descend for landing.

Our model describes a home runway (via homeStart and homeEnd), a destination runway (via destStart and destEnd), and our plane via its horizontal position (x), vertical position (y), horizontal direction (dx), vertical direction (dy), horizontal velocity (vx) and vertical velocity (vy).

We use a time-triggered model which is split into three stages of control and evolution: takeoff, inflight, and landing. In the takeoff phase, we start the plane on the home runway and require that it takes off into the air. We move to the inflight phase once the plane leaves the ground. The inflight phase governs everything that the plane can do until it needs to land – the phase ends once the plane reaches the ground. The landing phase begins once the plane is on the ground and controls how it can come to a stop. In each stage, we protect it with a guard and constrain the evolution domain such that the model is in the phase. This preserves the chronological order of a flight. As such, the model, at a high level can be summarized as:

```
(preconditions) ->
[{  takeoff controls
    {takeoff evolution & not taken off}
 }*
 ?(check that we have taken off);
 {
    inflight controls
    {inflight evolution & not landed}
 }*
 ?(check that we have landed);
 {
    landing controls
    {landing evolution}
  }*
](postconditions)
```



2.1 Iterative Design

In modelling the system, we choose to use an iterative design approach. Because our model can be split into three distinct phases, we are able to first create three simpler models – one for each phase. In this manner, we are able to solve simpler problems in each step. This is still a necessary component of the final model since with the use of nondeterministic repetition, any of the later phases may not run (i.e. for zero iterations), in which case we must still maintain the postcondition. In this manner we are also able to consider intermediate goals that the model must follow at various stages.

Once we have these separate models, we can combine them into a complete model of the plane. At this point, we will have additional considerations to make – in particular those pertaining to how the various phases interact with each other. We need to ensure that no phase will cause a guaranteed safety violation in a future phase; for example, the inflight phase cannot allow the plane to fly past the destination runway completely.

With a full model of the plane, we are then able to add further complexity to it to make it a more faithful representation of reality. This would involve loosening some of the assumptions that the original model relied on. In the current model, for example, we do not allow any drag during takeoff or thrust during landing; this would be somewhere that we can allow greater control options.

2.2 Custom Definitions

We introduce several custom definitions to de-clutter the model and make it easier to read. The derivations of the more complicated definitions are explained in subsequent sections. These are summarized as follows:

Definition	Meaning
`stationary(vx, vy)`	true iff the plane is stationary
`onGround(y)`	true iff the plane is on the ground (`y = 0`)
`onHomeRunway(x)`	true iff the plane's `x` position is within the home runway
`onDestRunway(x)`	true iff the plane's `x` position is within the destination runway
`liftGEQWeight(v, L)`	true iff the lift force, as a function of `v` and `L` is at least that of the weight of the plane
`canTakeOff(x, vx)`	true iff the plane can take off before the home runway ends, given its current position and velocity
`timeToStop(vx, B)`	the time it takes to reach zero horizontal velocity as a function of the applied braking force
`distToStop(vx, B)`	the distance needed to reach zero horizontal velocity given the applied braking force
`canStopBeforeEnd(x, vx, B)`	true iff the plane can stop before the end of the runway if braking force of `B` is applied for `T` time
`distTravel(v, a, t)`	the distance travelled in time `t` with initial velocity and constant acceleration

2.3 Safety and Postcondition

Post-condition: Our project focuses on the safety, rather than the efficiency (e.g. fuel consumption or flight time), of flight. The desired safety properties are formalized in dL in the post-condition of our model:

```
((stationary(vx,vy) -> (onHomeRunway(x) | onDestRunway(x)))
&
((!stationary(vx,vy) & onGround(y)) ->
    (onHomeRunway(x) & vy = 0 & (vx^2*L*C1 >= planeMass*g | canTakeoff(x, vx))) |
    (onDestRunway(x) & canStopBeforeEnd(x, vx, maxD)))
&
((!stationary(vx,vy) & !onGround(y)) -> true)
)
```

Here, `C1` is the constant in computing the lift force, which is given by `lift = L * v^2 * C1`, where `C1` is strictly positive.

The post-condition, in simple English, is the conjunction of all the following conditions (based on the logical desired safety property that: our plane will start and lift off on the home runway, and descend and slows down on the destination runway):

1. If the plane is stationary, it must be on a runway (either the home or destination runway).

2. If the plane is not stationary, but is on the ground, then at least one of the two must hold:

    a. It is on the home runway, with no vertical velocity, and one the the two holds:

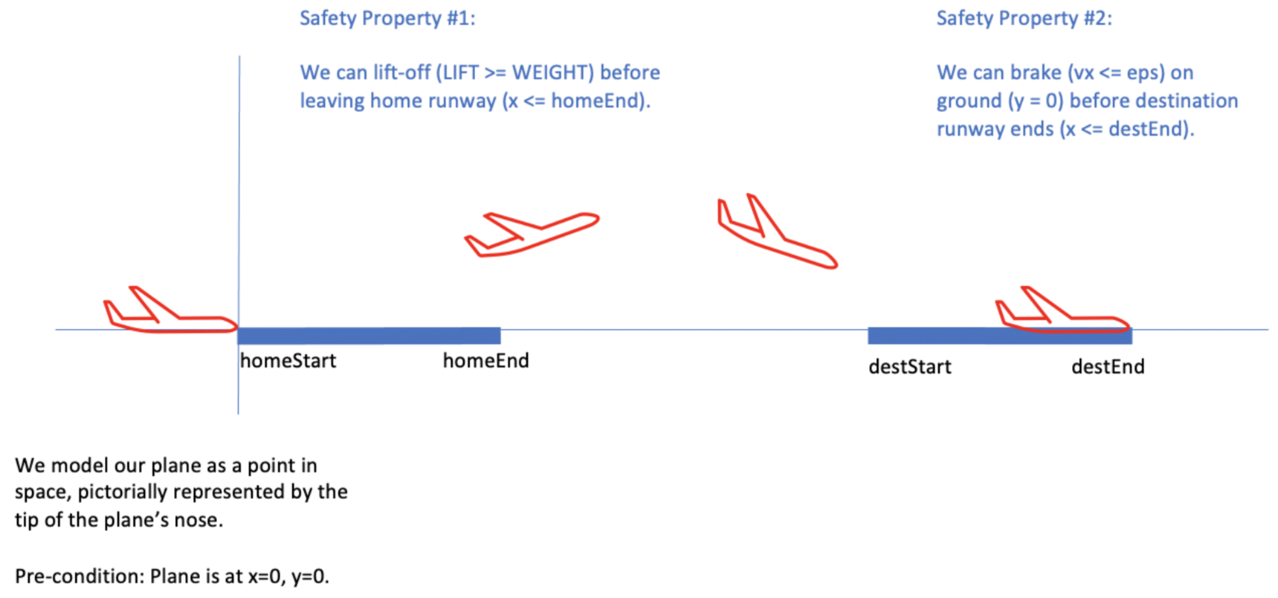
        i. lift is at least weight (so the plane can lift off the ground).

        ii. `canTakeOff` holds (so the plane can lift off before home runway ends).

    b. It is on the destination runway and `canStopBeforeEnd` holds (see Landing section for details of what `canStopBeforeEnd` entails).

3. If the plane is not stationary and not on the ground. We have not formalized the in-flight phase of the model, which this condition relies on. Informally it will be that the plane must not have crossed the destination runway end, and that the plane is able to descend onto the destination runway and brake before the destination runway ends.

Note that all the three conditions above (if-statements) are collectively exhaustive, ensuring the safety of the plane under all its states.


Safety Property #1:
We can lift-off (LIFT >= WEIGHT) before leaving home runway (x <= homeEnd).
Safety Property #2:
We can brake (vx <= eps) on ground (y = 0) before destination runway ends (x <= destEnd).
homeStart
homeEnd
destStart
destEnd
We model our plane as a point in space, pictorially represented by the tip of the plane's nose.
Pre-condition: Plane is at x=0, y=0.

3 Takeoff

As mentioned previously, we define the takeoff phase as the part when the plane is still on the ground. This ends when the plane leaves the ground, at which point, we switch to the inflight phase. Intuitively, this phase consists of the plane accelerating to gain enough speed such that the lift force is at least the weight – this would mean that the plane has enough lift to leave the ground.

3.1 ODE

In this phase, we follow the following ODE:

```
t' = 1,
x' = vx, y' = vy,
vx' = (dx * thrust)/planeMass,
vy' = 0,
dx' = 0, dy' = 0
&
t <= T & !liftGEQWeight(vx, L)
```

In our takeoff ODE, time evolves by 1, our x-position evolves by the horizontal velocity, vx, and the y position evolves with the vertical velocity, vy. We model the take-off phase as constant horizontal motion (no vertical motion), so dx'=0 and dy'=0 (i.e. the direction of our plane isn't changing in this stage). Lastly, vx' = (dx * thrust) / planeMass refers to that our horizontal velocity changes with our horizontal acceleration, which is calculated by computing the thrust in the horizontal direction dx * thrust and dividing this by the plane's mass. Finally, our take-off ODE's evolution domain restricts this evolution to when t <= T, the time trigger, and also that the lift force is strictly less than the weight force acting on the plane, when the plane begins to take off.

3.2 Control

The control of the model in this phase is as follows:

```
?(vx^2*L*C1 < planeMass*g);
t := 0;
thrust := *;
?(0 <= thrust & thrust <= maxThrust);

L := *;
?(0 <= L & L <= maxL);

if (!liftGEQWeight(vx, L)) {
    ?(
        (liftGEQWeight(vx + thrust/planeMass*T, L)
            & (x + distTravel(vx, thrust/planeMass, T) <= homeEnd))
    |
        (x + distTravel(vx, thrust/planeMass, T) <= homeEnd)
            & canTakeoff(x + distTravel(vx, thrust/planeMass, T), vx + thrust/planeMass*T)
    );
    { ODE }
}
```

We first allow the plane to pick any value of thrust and lift coefficient, within the constant bounds. In order to simplify the model, we only allow the drag coefficient chosen to be zero. In a successful takeoff, there is no reason that the plane should ever need to brake, given the abstractions we have already made. This, of course, is not an accurate representation of reality since planes may need to brake (e.g. an emergency abort of takeoff). We may choose to add such functionality in a future iteration.

We then have a main guard which ensures the safety of the chosen thrust and lift coefficient. In particular, there are three cases which we consider safe:

1. If the choice of lift coefficient is great enough that given the current velocity, the lift is at least the weight, then it is safe regardless of the thrust. Intuitively, this would mean that the plane accelerated to a velocity that's fast enough and then the pilot activated the flaps.

2. If at any point in the control cycle the lift exceeds the weight, then that means that we are able to take off. However, we must also ensure that we do not pass the end of the runway during this time. While we are able to make a more permitting guard computing the exact location which we can take off, this leads to unnecessarily complex equations. The current guard bounds the takeoff location.

3. The final case ensures that we are able to take off in some future control cycle. This means that after this cycle, we are still on the runway and given the location and velocity at the end of the cycle, it is still possible for us to take off safely.

3.3 Derivation of `canTakeoff`

First we define `canTakeoff` as follows:

```
    Bool canTakeoff(Real x, Real vx) <->
(maxThrust / planeMass * ((((g * planeMass) / (maxL * C1))^(1/2) - vx)
/ (maxThrust / planeMass))^2 / 2
+ vx * ((((g * planeMass) / (maxLiftCoeff * C1))^(1/2) - vx) / (maxThrust / planeMass))
+ x
<= homeEnd);
```

We want to be able to take-off safely. This means our plane should leave the ground (`y > 0`) before it exceeds the home runway (`x > homeEnd`). Our plane will leave the ground when lift is at least weight. Our condition is that lift is equal to weight by the time `x` is at the end of the runway. Since we know that `y=0, dx=1, dy=0`, and that we still have the option to set `vx' = maxThrust / planeMass` in the future.

Formulas for LIFT and WEIGHT are:

$$LIFT := vx^2 * maxL * C1.$$

$$WEIGHT := planeMass * g.$$

Rearranging, we get that for these to be equal, `vx = ((planeMass * g) / (mL * C1))^(1/2)`. Call this `vx` needed to lift-off, `vx_L`. So our condition is that we can reach `vx_L` by the time we reach the end of the runway.

Since we are able to in the future, use the most optimal settings, the time it takes to get from current `vx` to `vx_L` using `maxThrust` is `t = (vx_L - vx)/A`, where `A` is `maxThrust / planeMass`.

Distance needed to get from the current `vx` to `vx_L` is thus `d = A*t^2/2 + vx*t`. So, the `canTakeoff` condition is given by `d + x <= homeEnd`.

3.4 Proof Strategy

To prove this, we use the critical loop invariant of

```
dx^ 2 + dy^ 2 = 1 & (liftGEQWeight(vx, L) | canTakeoff(x, vx)) & vy=0 & y=0 &
onHomeRunway(x) & dx=1 & vx >= 0
```

The key components of this loop invariant are that

- The unit vector is always preserved – obviously this must hold
- Either lift is at least weight, or it `canTakeoff` – In the first case, we are at the cusp of taking off and so this allows the system to transition into the next phase; in the second case, we are still able to take off in the future
- Velocity and position vertically is zero – this must hold since it is how we define this phase. Likewise, with the condition that maintains that the unit vector is strictly horizontal
- The plane must be on the home runway – if we are not on the home runway, it is obviously unsafe
- horizontal velocity is positive – we do not allow the plane to travel backwards in this model

4 Landing

We define the landing phase as starting as soon as the plane touches the destination runway until it reaches a horizontal velocity of zero.

4.1 ODE

In this phase, we follow the following ODE:

```
t'=1,
x'= vx, y'= vy,
vx' = -B / planeMass,
vy' = 0,
dx'= 0, dy'= 0
&
t <= T & vx >= 0
```

In our landing ODE, time evolves by 1, our `x`-position evolves by the horizontal velocity, `vx`, and the `y` position evolves with the vertical velocity, `vy`. We model the landing phase as constant horizontal motion (no upward/ downward motion), so `dx'=0` and `dy'=0` (i.e. the direction of our plane isn't changing in this stage). Lastly, `vx' = -B / planeMass` refers to the fact that our horizontal velocity changes with our horizontal acceleration, which is calculated by the braking force applied (`B > 0`) divided by the plane's mass (`planeMass`). Finally, our take-off ODE's evolution domain restricts this evolution to when `t <= T` and also that `vx >= 0` (i.e. we never go backwards).

4.2 Control

Since we do not want the plane to lift above the ground in this phase, we set the lift coefficient to 0, and since speeding up in the horizontal direction does not help us eventually slow down to `eps`, we also set `thrust` to 0. This of course, is not true to reality, since we may want to speed up even on the destination runway (to reach the airport faster), but we can abstract that need for our purposes of safety. Since we set both the `liftCoeff` and `thrust` to 0, in the landing phase we effectively rely solely on our control of the braking force, `B`, to slow the plane down to horizontal velocity of zero. The control of the plane in the landing phase is modelled as follows:

```
t := 0;
vy := 0;
dy := 0;
dx := 1;

thrust := 0;
L := 0;
B := *;
?(0 < B & B <= maxB);

?(canStopBeforeEnd(x, vx, B));
```

We first discretely set the vertical components. This is because we allow a small negative (downwards) minimum landing velocity as we enter this phase. We allow this because it approximates the action of the normal force when hitting the ground.

We allow the plane to pick any value of the braking force, `B`, within its constant bounds. We do not allow thrust because to simplify the model – since thrust is the exact opposite of the braking force, this can be equivalently modelled with different constant bounds on `B`.

We then have a main guard (`canStopBeforeEnd`), which essentially checks if we can reach a horizontal velocity of zero before the end of the destination runway.

```
Real timeToStop(Real vx, Real B) = (vx / (B / planeMass));

Real distToStop(Real vx, Real B) = (vx^2 / (2 * (B / planeMass)));

Bool canStopBeforeEnd(Real x, Real vx, Real B) <->
    ((timeToStop(vx, B) <= T & x + distToStop(vx, B) <= destEnd)
    |
    (vx-B/planeMass*T >= 0
        & x + vx*T + (-B/planeMass)*T^2/2 + distToStop(vx - B*T/planeMass, maxB) <= destEnd));
```

`canStopBeforeEnd` essentially checks if any one of the following two conditions hold:

1. We are able to come to a stop within this control cycle and that that stop is before the end of the runway
2. We are not able to stop in this control cycle (`vx - B/planeMass*T >= 0`) and we are able to stop before the end of the runway in a future control cycle if we use the maximum braking force.

4.3 Proof Strategy

To prove this, we use the critical loop invariant of

`dx^2 + dy^2 = 1 & y=0 & vy=0 & onDestRunway(x) & dx=1 & ((vx = 0 & x <= destEnd) | canStopBeforeEnd(x, vx, maxB))`

The key components of this loop invariant are that

- The unit vector is always preserved – obviously this must hold
- Velocity and position vertically is zero – this must hold since it is how we define this phase. Likewise, with the condition that maintains that the unit vector is strictly horizontal
- The plane must be on the destination runway – if we are not on the home runway, it is obviously unsafe
- Either we are already stopped, in which case we are done running the model, or we can stop before the end.

5 In-flight

The inflight phase consists of after the plane takes off until the point that it returns to the ground. Throughout this phase, we must maintain that the plane does not reach the ground between the runways, it never gets too close to the ground that it is bound to crash, and that it still has enough time to land before the end of the runway. We summarize the motion with the ODEs as follows:

```
x'= vx, y'= vy,
vx' = dx * thrust / planeMass,
vy' = ((vx^2 + vy^2) * L * C1 - g * planeMass + dy * thrust) / planeMass,
dx'= vx / (vx^2 + vy^2)^(1/2),
dy'= vy / (vx^2 + vy^2)^(1/2)
&
t <= T & vy >= minLandingVelocity
```

Where `minLandingVelocity` is a negative (downwards) constant of the maximum (magnitude) of velocity the plane can have when it lands. The unit vector evolves as the component of the velocity that the plane travels in. In the horizontal direction, `vx'` evolves proportionally to the horizontal component of the thrust. In the vertical direction, we must account for the vertical component of thrust, the weight of the plane from gravity, and the lift, which is proportional to the square of velocity.

This phase was far more complex than take-off or landing, primarily because of the complexities of aerodynamics. Even when we tried simplifying our model to not include drag, the complex ODE of thrust (dependent on velocity squared) was far too difficult to come up with simple, provable bounds for. In the future, we would possibly even further break down the stages of in-flight and enforce some fixed controller actions.

6 Conclusion

We have proposed a provably safe model for both takeoff and landing of aircraft. We have shown the key aspects of modelling an aircraft; these models can act as the groundwork for more robust models in the future. In particular, the main avenues of potential future work would be to complete the model and proof of the in-flight phase. Additionally, it would seek to eliminate some of the simplifications from reality that the models presented have made.

7 Summary of Deliverables

- This comprehensive report of the project
- Model of takeoff: model successfully completed and proved, relevant files submitted
- Model of landing: model successfully completed and proved, relevant files submitted
- Model of in-flight: core ideas explained in writeup

8 Member Contribution

Majority of modelling and proving were worked on synchronously and contribution was equal. Writeup of checkpoint and final were split equally.

References

[1] Dunbar, Brian. "Small Aircraft Transportation System." NASA, NASA, www.nasa.gov/centers/langley/news/factsheets/SATS.html.
[2] "Future Flight: A Review of the Small Aircraft Transportation System Concept." Future Flight: A Review of the Small Aircraft Transportation System Concept | Blurbs New | Blurbs | Main, www.trb.org/Main/Blurbs/153338.aspx.
[3] "Cessna 172." Wikipedia, Wikimedia Foundation, 26 Sept. 2020, en.wikipedia.org/wiki/Cessna_172.
[4] Boeing Airplane Company. Statistical Summary of Commercial Jet Airplane Accidents 1959-2018. Boeing Commercial Airplane Group, 2019.
[5] "172AVIONES COLOMBIA - C172 L1P L." Doc8643, doc8643.com/aircraft/C172.
[6] "AOPA Asf/Publications/inst_reports2.Cfm." Air Safety Institute Instructor Reports, 4 Mar. 2016, www.aopa.org/asf/publications/inst_reports2.cfm?article=4567.
[7] https://www.faa.gov/about/office_org/headquarters_offices/ang/offices/tc/about/campus/faa_host/labs/simulation_team/tgf/media/AircraftDynamicsModel.pdf
[8] http://eprints.whiterose.ac.uk/134664/1/Paper.465Rv.pdf
[9] http://www.csl.sri.com/users/rushby/papers/csl-93-7.pdf
[10] https://www.researchgate.net/publication/240944797_Formal_Verification_of_Flight_Critical_Software
[11] https://www.cs.cmu.edu/~aplatzer/pub/RCAS.pdf
[12] https://www.nasa.gov/pdf/601262main_ModelingFlight-ebook.pdf