Modelling the trajectory of a skydiver

Group 7

September 01, 2013

Philipp Müller
Faculty of Engineering Sciences, Tampere University of Technology,
P.O. Box 692, FI-33101 Tampere, Finland

Dario Paccagnan
DTU Compute, Technical University of Denmark,
Matematiktorvet Building 303 B, DK-2800 Kgs. Lyngby, Denmark

Kevin Polisano
Engineering School, Ensimag,
BP 72,38402 Saint Martin D'Heres, France

Katharina Rafetseder
Institut für Mathematik, Johannes Kepler Universität Linz,
Altenberger Straße 69, 4040 Linz, Austria

Miriam Ruiz Ferrández
Faculty of Mathematics, University of Santiago de Compostela,
Rúa Lope Gómez de Marzoa, Campus sur 15782, Santiago de Compostela, Spain

Tom Slenders
Faculty of Mathematics and Computer Science, Eindhoven University of Technology,
Den Dolech 2, Postbus 513, 5600 MB Eindhoven, The Netherlands

Katherine Tant
"Mathematics and Statistics", University of Strathclyde,
LT1021, Livingston Tower, 26 Richmond Street, Glasgow, G1 1XH , Scotland

Angelos Toytziaridis
Faculty of Engineering, University of Lund,
Box 117, SE-221 00 Lund, Sweden

Instructor: Kshitij Kulshreshtha
Institut für Mathematik, Universität Paderborn,
Warburger Straße 100, 33098 Paderborn, Germany

Abstract

Skydiving is the extreme sport of exiting an aircraft mid-flight and free-falling, before releasing a parachute on the final approach to Earth. If the parachute is not deployed in time, the diver risks suffering from severe, or even fatal, injuries. The need to manage this risk has generated a market for Automatic Activation Devices (AADs), which will automatically release a parachute when it is detected that the diver is in danger. CYPRES is such a device created by Airtec. This poject focusses on advancing the methodology behind CYPRES in the hopes of improving its ability to detect when the diver breaches a safety threshhold based on height and acceleration measurements. An Extended Kalman Filter, fusing measured data with an ODE based on the physics of free-fall, has resulted in a predictive model, approximating the time left until a parachute must be deployed. With some further development, this approach could feasibly further minimise the risk associated with the sport.

1 Introduction

In recent years, the extreme sport of skydiving has become increasingly popular. The risk associated with the sport is high, with an estimation of over 100 fatalities occurring since 2011 [**?**]. The need to manage this danger has generated a market for tools such as Automatic Activation Devices (AADs). These AADs automatically deploy a reserve parachute when necessary, thus reducing the number of fatal accidents caused by the diver failing to 'pull' or the parachute failing to deploy.
One such device is the CYPRES (Cybernetic Parachute Release System) produced by Airtec. CYPRES determines the height above sea level by measuring the barometric pressure[1]. If the diver's velocity exceeds a predetermined value at a preset altitude, it will automatically release a reserve parachute.
CYPRES is available in several models such as Expert, Tandem, Student and Speed, all of which differ only in the preset parameters for velocity and height, allowing the diver to choose the most suitable level for his needs and experience.

Type of diver	Critical height	Critical velocity
Experts	$225\,m$	$35\,\frac{m}{s}$
Students	$300\,m$	$13\,\frac{m}{s}$
Tandem	$580\,m$	$35\,\frac{m}{s}$
Speed	100-$225\,m$	$43\,\frac{m}{s}$

Clearly the height estimation must be as precise as possible, but due to the changing position and posture of the diver, the measured pressure is very noisy and consequently, there exists a margin of error for the estimated height.
In order to cope with this difficulty Airtec wants to improve CYPRES by including measurements (taken every $0.25s$) coming from an accelerometer attached to the skydiver.

The objective of the project is to fuse the data coming from the accelerometer with that from the barometric probe in order to enhance the precision of CYPRES. A prediction for the future height is also desirable.

[1] Once the barometric pressure P is known, it is possible to compute the height z above the sea level with the following relation $\frac{P}{P_0} = \left(\frac{T_0}{T_0 + L_0 z}\right)^{\frac{gM}{R^* L_0}}$, where P_0 and T_0 are the barometric pressure and the standard temperature at sea level, L_0 is the standard temperature laps rate, M the molar mass of dry air and R the universal gas constant.

Approach

The core tool developed in this project report is an extended Kalman filter (EKF), which fuses the information from the sensors with the knowledge of the mathematical model governed by the physics.
As is well known, the EKF allows one to enhance the precision with respect to both the measurements and mathematical model. Consequently, the first important step is to determine the model of the physical phenomenon we want to study. Describing the dynamic from the take off of the aeroplane to the diver's point of exit is difficult, since aerodynamic forces play an important role, and are irrelevant in regards to the operation of the AAD. It was therefore decided appropriate to model the free-falling phase only, i.e. the period of time between the jump from the plane and the parachute opening. The EKF starts when the skydiver jumps, hence we needed to automatically determine the specific instant when the jump occurred and the initial conditions.

In section 2 we construct the mathematical model with particular attention to the estimation of unknown parameters involved in the equations. Section 3 is devoted to automatic determination of the jumping point and the initial conditions. The EKF and its implementation are then presented in section 4. In the last part we present the results we have obtained using real world data and some conclusions.

2 Model of a free-fall skydiver

We want to model the dynamic of a free-falling object of mass m. In particular, we are interested in its vertical position z. For this purpose we project Newton's second law on the vertical axis and consider all the forces F_z acting on that direction

$$m\,\ddot{z} = F_z\ , \tag{1}$$

where $\ddot{z}$ is the vertical acceleration.
Within the model, gravity and wind drag must be taken into account.

- The gravity force is given by $G = -mg$, where we can assume the gravitational acceleration $g = 9.80665\frac{m}{s^2}$ being constant as it changes significantly only above an altitude of 100 kilometres.
- The drag force has a more complex structure, which can be expressed

compactly using the drag equation as

$$D = \frac{1}{2}\rho(z)\dot{z}^2 c_D A \ ,$$

where $\dot{z}$ is the vertical speed, c_D the dimensionless drag coefficient and A is the horizontal cross sectional area of the diver.
The mass density of air $\rho(z)$ changes significantly with height according to the well known equation

$$\rho(z) = \rho_0 \left(\frac{T_0 + L_0 z}{T_0}\right)^{(-\frac{gM}{R^* L_0})-1} .$$

In the previous formula T_0 is the temperature at sea level, $L_0 = -0.0065\,\frac{K}{m}$ is the standard temperature lapse rate, $M = 0.0289644\,\frac{kg}{mol}$ is the molar mass of air, $R^* = 8.31432$ the universal gas constant, and ρ_0 the mass density at sea level. In later calculations, T_0 and ρ_0 will be set $T_0 = 293\,K$ and $\rho_0 = 1.2041\frac{kg}{m^3}$.

The major difficulty comes from c_D and A as they change during the jump depending on the skydiver's posture, which is unknown. We therefore define the coefficient $c^*(t) := c_D A$ which is dependent on time.
In the following, we will only consider the time average value of $c^*(t)$, which is clearly a constant. In the next section we are going to present a non-linear estimation technique that allows us to determine the best time average value c^* at the present time based on all previous measurements.

If we substitute the specific expressions for the gravity force and wind drag in (1), we get the final form of the ODE

$$\ddot{z} = -g + \frac{1}{2m}\rho(z)\dot{z}^2 \underbrace{c_D A}_{=:c^*(t)}, \tag{2}$$

with $\rho(z) = \rho_0 \left(\frac{T_0 + L_0 z}{T_0}\right)^{(-\frac{gM}{R^* L_0})-1}$.

Estimation of c^*

The purpose of this section is to introduce a non-linear fitting technique that allows us to determine the best time average value c^*, given previous

measurements from the free-fall phase.
The first step is to rewrite the ODE as a system of first order equations

$$\begin{cases} \dfrac{dz}{dt} = v \\ \dfrac{dv}{dt} = -g + \dfrac{1}{2m}\rho(z)v^2c^*. \end{cases}$$

Given data points $(t_i, y_i)_{i=1}^k = (t_i, z_i)_{i=1}^k$ up to timestep t_k, we compute the estimate of the parameter c^* as the solution to the following minimization problem

$$\begin{cases} \min\limits_{c\in\mathbb{R}} & \phi = \dfrac{1}{2}\displaystyle\sum_{i=1}^{m} \|\hat{y}(t_i) - y_i\|_2^2 + \dfrac{1}{2} \underbrace{\alpha\|c\|_*^2}_{\text{regularization}} \\ s.t. & \dfrac{d\boldsymbol{x}}{dt}(t) = f(t, \boldsymbol{x}(t), c) \quad \boldsymbol{x}(t_0) = \boldsymbol{x}_0 \\ & \hat{y}(t) = g(\boldsymbol{x}(t), c) \\ & c_l \le c \le c_u, \end{cases} \tag{3}$$

where in our case

$$\boldsymbol{x}(t) = \begin{bmatrix} z(t) \\ v(t) \end{bmatrix}, \qquad f(t, x(t), c) = \begin{bmatrix} v(t) \\ -g + \frac{1}{2m}\rho(z)v(t)^2c \end{bmatrix}, \qquad g(\boldsymbol{x}(t), c) = z(t).$$

The idea for determining c^* is to find the value of c that minimizes the absolute difference between measured values, y_i and analytical values, $\hat{y}(t_i)$. The term $\alpha\|c\|_*^2$ in (3) is necessary to guarantee the well-posedness of the general problem. However, since we are looking for a parameter c^*, which is a time average and thus a constant over time, α can be chosen as zero. More information about regularization methods can be found in [5].
To solve the problem we need to know the initial condition, i.e. the height $z(t_0)$ and the velocity $v(t_0)$ at the jumping time. The identification of the jumping instant and the estimation of the initial condition is the topic of the next section.

3 Jumping time detection and initial conditions

To continue with the development of the ODE model it is necessary to devise an automatic technique to find the time at which the diver jumps from the

aircraft, so as to calculate the velocity for the initial conditions. The presence of noise in the data (pressure, acceleration) provided by Airtec makes this quite difficult.
The first attempt to extract this value involved looking at the z component of the acceleration vector. Examining the unfiltered data (see Figure 1) a large spike is observed after approximately 5000 time steps, i.e. after approximately 1250 seconds. However, noise prevents us from automatically choosing the jump point from this plot.

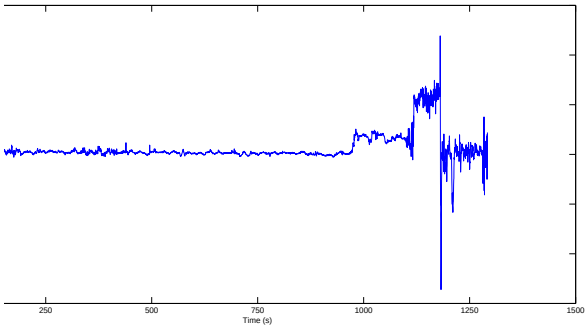

Figure 1: Vertical component of the acceleration at every time step.

To accentuate the drop in acceleration at the jump point, differences between neighbouring points were taken, allowing for analysis of the rate of change in acceleration over time (see Figure 2). A median filter[2] was then applied to this data (Figure 3), and the jumping point was selected to correspond to the maximum value in the smoothed derivative.
Of the six data sets provided, this approach was successful in five. The uncertainty in the sixth case motivated a change in tactic.

[2]The filters we applied are discussed in subsection 3.1 and 3.2.

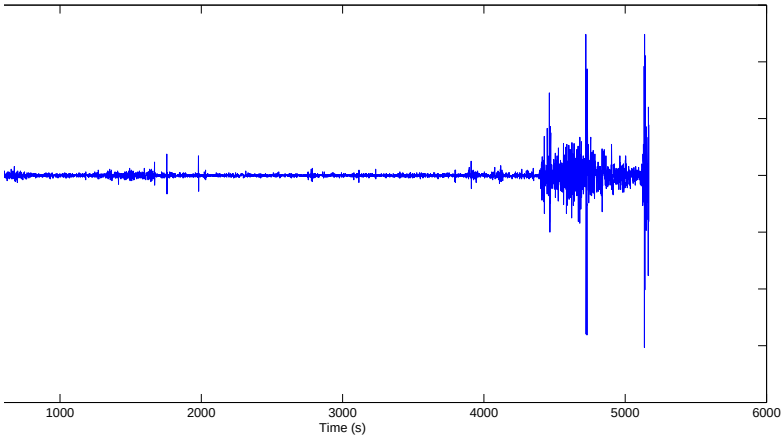

Figure 2: Derivative of vertical acceleration component.

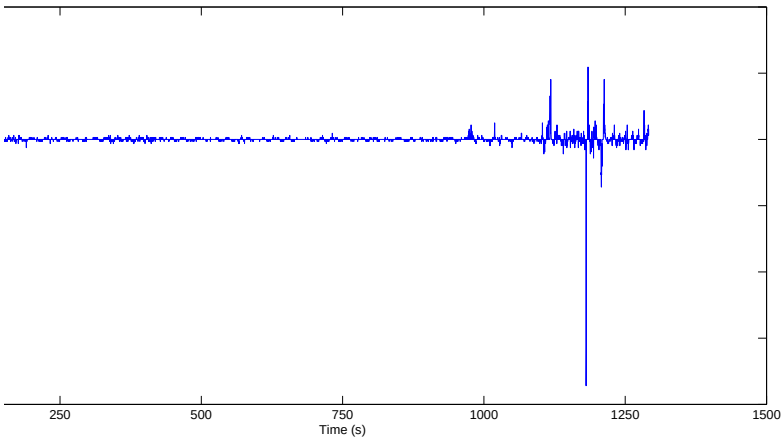

Figure 3: Filtered derivative of vertical acceleration component.

Instead of considering only the z component of the acceleration vector, it was decided to examine the vector norm. Figure 4 depicts the differentiated unfiltered and filtered acceleration vector norms. In this case the median filter with a window of size 21 was applied. Note that as the filter uses a symmetric window, it is necessary to know what values occur in 10 future data points. This gives a delay of $10 \times 0.25\,s = 2.5\,s$ in determining the jump point. The delay is acceptable in relation to the total length of the free-fall, which lasts approximately one minute. The final algorithm uses a threshold (approximately 1/2 of the mean value of the dataset) to estimate the time

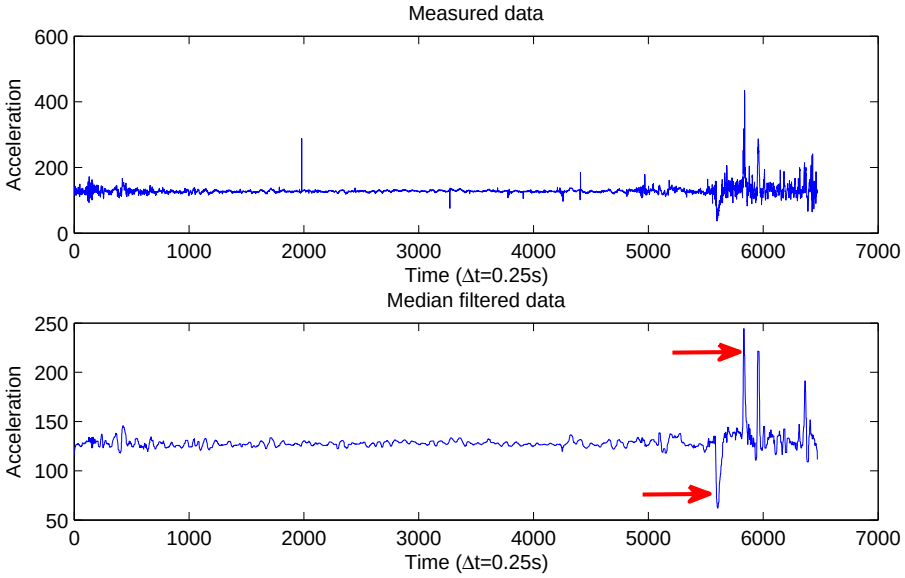

Figure 4: The median filter can be used to determine the jump point and parachute opening point. The acceleration is the norm of the three axis.

at which this occurs. The first point below this threshold is taken to be the jump point. It must be noted that the median filter window size utilized in the algorithm was never optimized and there is potential to further minimize the current delay of $2.5s$.

Beside the initial height the initial velocity is needed. The idea is to calculate the initial velocity by means of a central difference quotient, i.e.

$$v(t) = \frac{h(t + \Delta t) - h(t - \Delta t)}{2\Delta t},$$

where $\Delta t = 0.25$.
A moving average filter was used to smooth the height measurements. However, by considering the green filtered height in figure 5, it can be observed that the numerical velocity (calculated via the central difference quotient) changes considerably around the jump point. Hence, we add a delay (15 samples ≈ 3.75 sec.) to the determined jump point and calculate the initial velocity at this time. In the following computations we use this delayed jump point to calculate the initial conditions for our ODE.

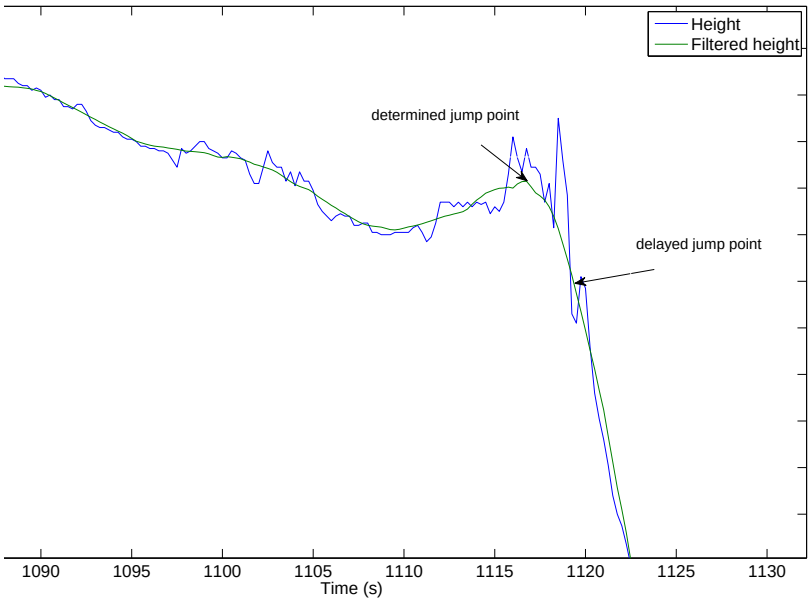

Figure 5: Measured height compared with filtered height.

3.1 Median Filter

As mentioned above, the median filter was applied to the acceleration data. For each time step t_i a window of size n is taken, centered on that point. The values lying within the window are sorted into ascending order and the median is taken. This value is then assigned to t_i and the window is shifted by one time step to the right where the process is reiterated. Median filters are typically used because of their stability with respect to outliers.

$$\hat{t} = median(sort(t_{i-n/2}...t_i...t_{i+n/2})) \tag{4}$$

3.2 Moving average filter

The moving average (MA) filter averages different subsets of the data to give a smoothed approximation of the measured signal. In its simplest form, $\hat{t}_i$ is the unweighted mean of n previous data points

$$\hat{t}_i = \frac{t_i + t_{i-1} + ... + t_{i-n+1}}{n} \tag{5}$$

This process is carried out for every point in the data set, excluding the first $(n-1)$ points which use only the previous data available. More advanced MA filters are available to improve the approximation however for the purposes of this project, the simple MA filter was sufficient.

4 Extended Kalman filter

So far, we have constructed a model (ODE) of a free-falling skydiver and devised a technique to find the jumping time at which we calculate the initial condition for the differential equation.
We are ready to introduce the core tool: the extended Kalman Filter (see e.g. [3, p. 195 ff.], [4, p. 273 ff.]). The well-known Kalman filter (see e.g. [6, p. 206 ff.]) is an algorithm that fuses together measured data coming from a physical phenomenon and the (linear) model governing it. The output is a more precise estimate than what each of these two can yield alone. The EKF is the non-linear extension of the Kalman filter.
In this specific case we want to fuse together the measured pressure and acceleration data with the physical model we have constructed (ODE).

We begin by writing our ODE as a system of first order ODEs:

$$\frac{d^2z}{dt^2} = \frac{D}{m} - g \Longleftrightarrow \begin{cases} \dot{z} = v & z(t_0) = z_0 \\ \dot{v} = \frac{1}{2m}\varrho v^2 c_D A - g & v(t_0) = v_0. \end{cases} \tag{6}$$

We let $\boldsymbol{X} = \begin{pmatrix} z \\ v \end{pmatrix}$, thus the equation can be written as

$$\dot{\boldsymbol{X}} = \mathfrak{F}(\boldsymbol{X}) \quad \text{where } \mathfrak{F}\begin{pmatrix} z \\ v \end{pmatrix} = \begin{pmatrix} v \\ \frac{1}{2m}\varrho v^2 c_D A - g \end{pmatrix}.$$

Here v corresponds to the 'speed' of the skydiver which is assumed to be mainly oriented towards Earth.
The first step is to transform the system of differential equations previously presented into a discrete dynamic and to add noise as follows

$$\begin{aligned} \boldsymbol{x}_k &= \begin{bmatrix} z_k \\ v_k \end{bmatrix} = f(\boldsymbol{x}_{k-1}) + \boldsymbol{w}_{k-1} \\ \boldsymbol{y}_k &= \begin{bmatrix} z_k \\ \dot{v}_k \end{bmatrix} = c(\boldsymbol{x}_k) + \boldsymbol{u}_k, \end{aligned}$$

where $f(\boldsymbol{x}_k)$ is called the state function, and $c(\boldsymbol{x}_k)$ observation function. Both functions f and c can be non-linear but must be differentiable to enable us to use the EKF. Furthermore, $\boldsymbol{u}_k$ and $\boldsymbol{w}_k$ are the state transition and observation noises, which are both assumed to be zero mean multivariate Gaussian noises with covariance matrices $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$ respectively. In our case we choose $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$ to be constant. To find an accurate $\boldsymbol{Q}_k$ a large database of samples should be analysed. However, due to lack of prior data, we were forced to make some assumptions. We chose $\boldsymbol{Q}_k = \begin{bmatrix} 5^2 & 0 \\ 0 & 0.2^2 \end{bmatrix}$ where 5 and 0.2 relate to estimations of the standard deviation of the height and velocity respectively.
The standard deviation of the height measurements $\sigma = 15$ m was provided by Airtec, allowing us to make an informed estimate for $\boldsymbol{R}_k = \begin{bmatrix} 15^2 & 0 \\ 0 & 0.5^2 \end{bmatrix}$.

- The state function f is found by discretizing the model (6). We choose to do this by applying a fourth order Runge-Kutta method since it is much more stable and more accurate than, for example, the Euler method. The fourth order Runge-Kutta for a step size $h > 0$ is defined as:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) = f(\mathbf{x}_k)$$

with

$$\begin{cases} k_1 = \mathfrak{F}(\mathbf{x}_k) \\ k_2 = \mathfrak{F}\left(\mathbf{x}_k + \frac{h}{2}k_1\right) \\ k_3 = \mathfrak{F}\left(\mathbf{x}_k + \frac{h}{2}k_2\right) \\ k_4 = \mathfrak{F}(\mathbf{x}_k + hk_3) \end{cases}$$

- Each observation consists of the couple (height, acceleration), thus the observation function is given by

$$c(z, v) = \begin{pmatrix} z \\ a \end{pmatrix} = \begin{pmatrix} z \\ \dot{v} \end{pmatrix} = \begin{pmatrix} z \\ \frac{1}{2m}\varrho v^2 c_D A - g \end{pmatrix}.$$

The main idea of the EKF is to linearize state and measurement functions around an estimate of the current mean value and its covariance, wherefore it uses the Jacobian matrix of f and c at a point $\boldsymbol{x}_k$, which we denote as $\boldsymbol{F}_k$ and $\boldsymbol{C}_k$ respectively. By definition

$$\boldsymbol{C}_k = \begin{pmatrix} \frac{\partial f_1}{\partial z}(z_k, v_k) & \frac{\partial f_1}{\partial v}(z_k, v_k) \\ \frac{\partial f_2}{\partial z}(z_k, v_k) & \frac{\partial f_2}{\partial v}(z_k, v_k) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{1}{2m}\rho'(z_k)v_k^2 c^\star & \frac{1}{m}\rho(z_k)v_k c^\star \end{pmatrix}.$$

Determining the Jacobian matrix of f by hand is hard because f is a composition of $\mathfrak{F}$ four times with itself, so we would need to apply the chain rule. Fortunately, there exists a tool to evaluate the derivative of a function specified by a computer program, called "algorithmic differentiation". For this purpose, we used the Matlab program "ADiMat".

The extended Kalman filter is defined by the following set of recursion [1], which can be divided into two parts: predicting the current state using information from the previous state, and updating this state estimate using measurements/observations of the current time step.

Prediction

Predicted state estimate	$\hat{\boldsymbol{x}}_{k\|k-1} = f(\hat{\boldsymbol{x}}_{k-1\|k-1})$
Predicted covariance estimate	$\boldsymbol{P}_{k\|k-1} = \boldsymbol{F}_{k-1}\boldsymbol{P}_{k-1\|k-1}\boldsymbol{F}_{k-1}^T + \boldsymbol{Q}_{k-1}$

Update

Innovation or measurement residual	$\tilde{\boldsymbol{y}_k} = \boldsymbol{y}_k - c(\hat{\boldsymbol{x}}_{k\|k-1})$
Innovation (or residual) covariance	$\boldsymbol{S}_k = \boldsymbol{C}_k\boldsymbol{P}_{k\|k-1}\boldsymbol{C}_k^T + \boldsymbol{R}_k$
Near-optimal Kalman gain	$\boldsymbol{K} = \boldsymbol{P}_{k\|k-1}\boldsymbol{C}_k^T\boldsymbol{S}_k^{-1}$
Updated state estimate	$\hat{\boldsymbol{x}}_{k\|k} = \hat{\boldsymbol{x}}_{k\|k-1} + \boldsymbol{K}_k\tilde{\boldsymbol{y}}_k$
Updated estimate covariance	$\boldsymbol{P}_{k\|k} = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{C}_k)\boldsymbol{P}_{k\|k-1}$.

The state transition and observation matrices are defined by

$$\boldsymbol{F}_{k-1} = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k-1|k-1}}$$

$$\boldsymbol{C}_k = \left.\frac{\partial c}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k-1}}$$

To initialize the recursion, the initial state $\boldsymbol{x}_0$ is set to z_0, the altitude of the skydiver when he is jumping, and v_0 their velocity at that point as described in the previous sections.
The algorithm gives at each step an improved estimate called $\hat{\boldsymbol{x}}_{k|k}$, but also a future prediction of how the state would evolve. The l steps ahead prediction is obtained by applying $\hat{\boldsymbol{x}}_{k|k-1} = f(\hat{\boldsymbol{x}}_{k-1|k-1})$ l times. The future prediction is valuable as it allows us to compute the time left until the parachute must open automatically (i.e. until the height drops under the safety level).

5 Results and Conclusion

We first present the comparison for the height estimation in Figure 6. The ODE's analytical solution, the measured data, and the EKF's estimates are close to each other. In more detail, Figure 7 shows that the EKF prediction is affected - as designed - by the incoming measurements, while this is not true for the solution of the ODE.

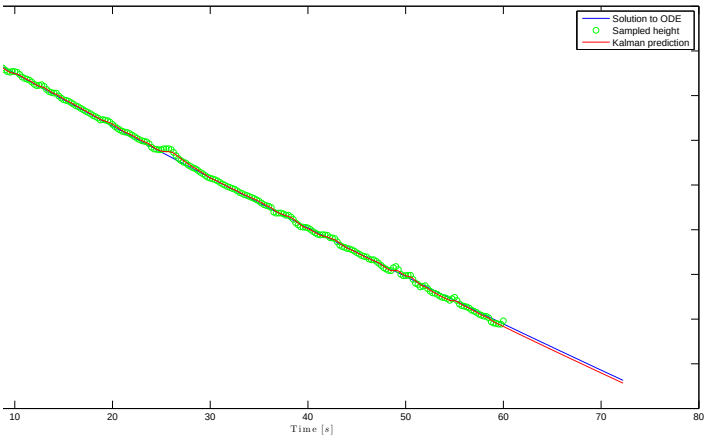

Figure 6: The change of height over total period of time.

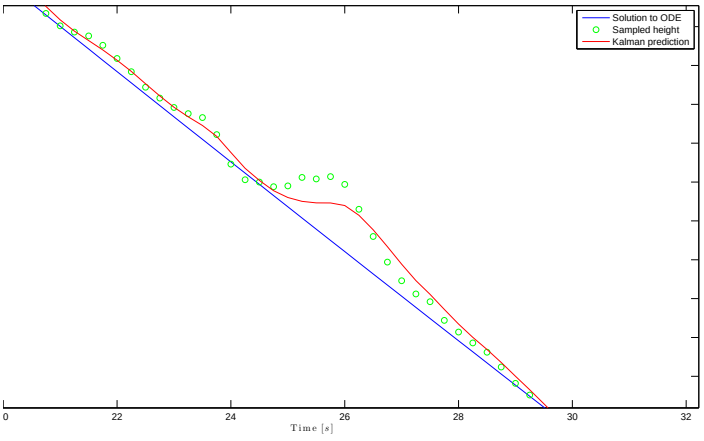

Figure 7: Comparison of sampled height, ODE solution and Kalman prediction

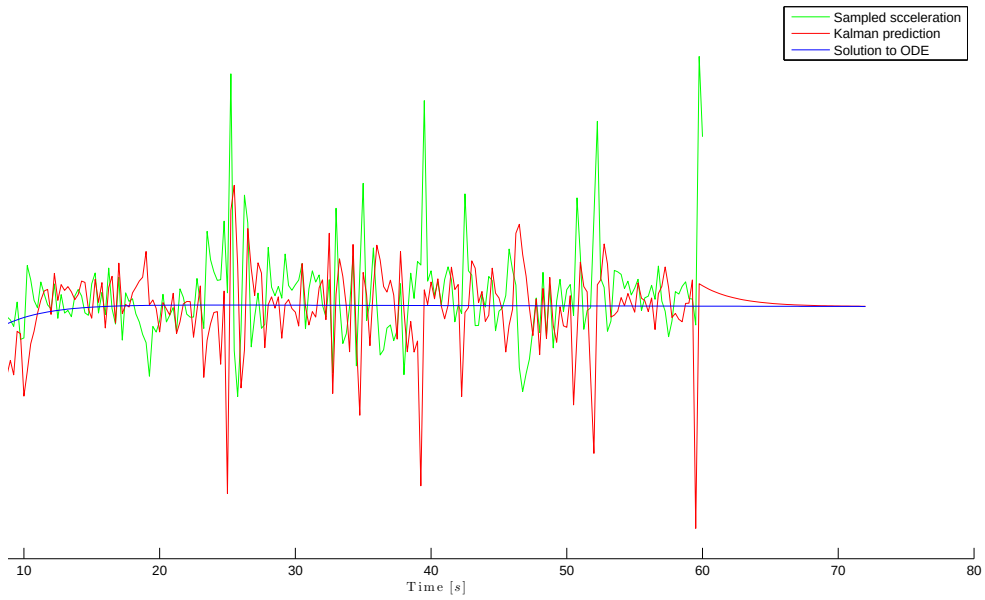

Figure 8: Comparison of sampled acceleration, solution to ODE and Kalman prediction

Figure 8 shows the predicted acceleration compared to the sampled acceleration and the solution of the ODE model. Although the values do not fit exactly one can say that the three curves have the same trend.

The vertical velocity is shown in Figure 9, where the red line represents the values coming from the Kalman prediction, while the blue one is the solution of the ODE. Even though they do not overlap exactly, they are still close to each other and begin to converge as time progresses. This is due to the way we have implemented our algorithm. The EKF is based on the theoretical model (6) whose solution depends on the parameter c^*. To enhance the accuracy we update c^* every 5 samples in our EKF algorithm. As we get more observations, the updated c^* gives a better estimation of the model and the Kalman filter works more effectively.[3]

The most valuable result for Airtec is represented in Figure 10, which depicts the predicted time until automatic parachute release. Note that it does not reach zero as the parachute wass opened before automatic deployment was

[3]We have taken into account the computational load required for this extra loop and notice that each single c^* evaluation costs only $0.25s$.

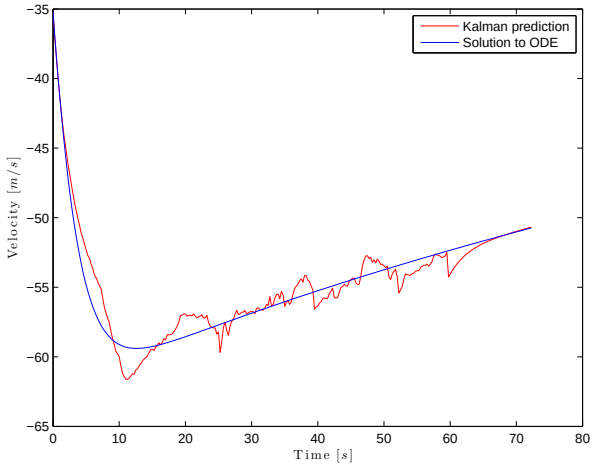

Figure 9: Velocity given as solution to ODE and Kalman prediction.

necessary.

A general idea for improvement would be to have two sensors, for example one on the front and a second on the back of the diver, and then take the mean value of the measurements. This could potentially reduce the noise resulting from acting wind and make the predictions more accurate.

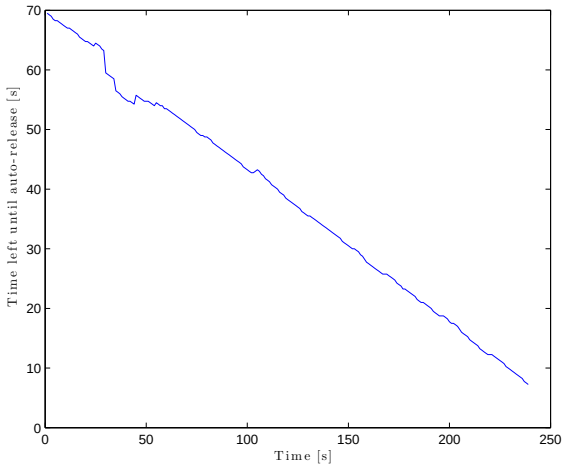

Figure 10: Predicted time left until parachute release given the samples t_k.

References

[1]

[2] Broken stick regression.

[3] Brian D. O. Anderson and John B. Moore. *Optimal Filtering*. Prentice-Hall information and system sciences. Dover Publications, Inc., 2005.

[4] Andrew H. Jazwinski. *Stochastic Processes and Filtering Theory*, volume 64 of *Mathematics in Science and Engineering*. Academic Press, 1970.

[5] Barbara Kaltenbacher, Andreas Neubauer, and Otmar Scherzer. *Iterative regularization methods for nonlinear ill-posed problems*, volume 6. Walter de Gruyter, 2008.

[6] Peter S Maybeck. *Stochastic Models, Estimation, and Control Volume 1*. Mathematics in Science and Engineering. Academic Press, 1979.

A Linear regression of velocity

Aside from the methods described above we also tried to apply a linear regression model to fit our data. It appeared that this method was less effective, therefore we chose not to use it. Nonetheless we add it here for possible further research.

Linear regression is a way to determine a linear function through some data points. Applying this to our model does not work, since it's obviously not linear. Therefore we chose to apply piecewise linear regression which does more or less the same thing, but only considers the data between some set breakpoints B. It finds a linear function between consecutive breakpoints and creates an overall continuous function.

For the linear regression we simply use this algorithm [2] and others exist as well. The main problem is finding the break points. Doing this manually gives very good results as can be seen in Figure 11. But we have to automate this in the real system. We tried a few approaches for this. Each of these started by applying a median filter.

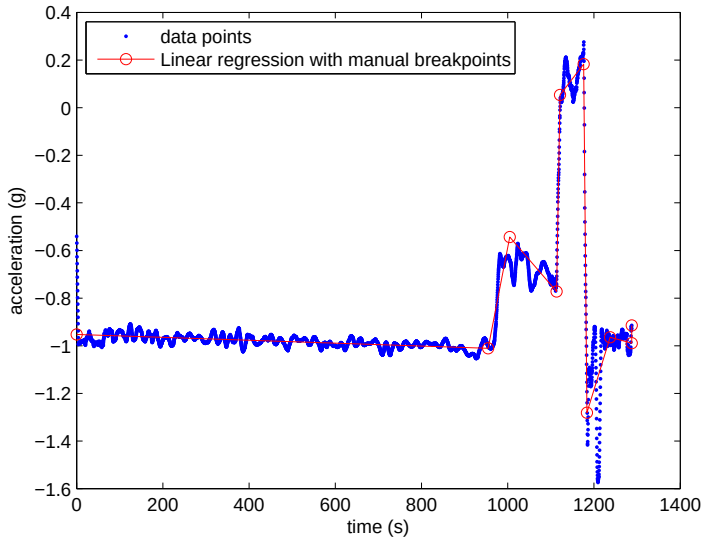

Figure 11: The linear regression found by manually picking breakpoints.

A.1 Median threshold filter

As seen in section 3.1 we can use two threshold values to find the jump point and parachute opening point. This idea can be extended by taking a lower threshold and taking all values that exceed the threshold. We are trying to estimate the acceleration function, this means that we can't look solely at the value of each point. We have to take a derivative or even a second derivative. Both seem to give results as can be seen in Figure 12.

The amount of points that result from this method vary a lot and therefore we also try to find the threshold automatically by setting the amount of points p we want and then automatically taking the p absolute highest values. Unfortunately most points that were founded were very close to each other if a low p was chosen. If a high p was chosen several interesting points are found, but it how good these points are differs a lot for different data sets. Therefore this method did not work as good as the previous method.

A.2 Postprocessing

After determining some interesting point by setting a threshold we find several points that are too close to each other. We apply to filters on them to eliminate the points that are too close together.

The first filter is a middle point filter. This filter tries to determine if several interesting points are close together and removes the middle points, but lets both left and right point exist. A certain window w must be chosen. The algorithm is shown in Algorithm 1. The reason for this algorithm is that a lot of points that have strange behavior are close together. Most of the time you only want the two exterior points for the piecewise linear regression as the others were probably created by noise.

Algorithm 1 Middle point filter(w, D)

Require: $w \in \mathbb{N}$, D a set of breakpoints

found = false
for $i \in \{2, \ldots, |D|\}$ **do**
  if found=false **then**
    if $D(i) - D(i-1) \leq w$ **then**
      found:=true
    end if
    $R(i) := D(i)$
  else
    if $D(i) - D(i-1) > w$ **then**
      $R(i) := D(i)$
      found:=false
    end if
  end if
end for
$R(1) := D(1), R(\text{length}(R) + 1) := D(|D|)$

We also use a second postprocessing algorithm called the close points filter to find two points that are close together. More than two points that are close together will be removed by the middle point filter, but if you have two points that are close together, the close points filter will resolve this. We choose a window w. Any two points that are at most w steps away from each other will be merged in a new point. This point will be the average of the two points. We apply the middle point filter first and the close points filter thereafter. This means that as long as we choose the window of the middle point filter as large as the window for the close point filter, there will never be more than two points close together.

A.3 Applying the filters

The resulting steps are as follows:

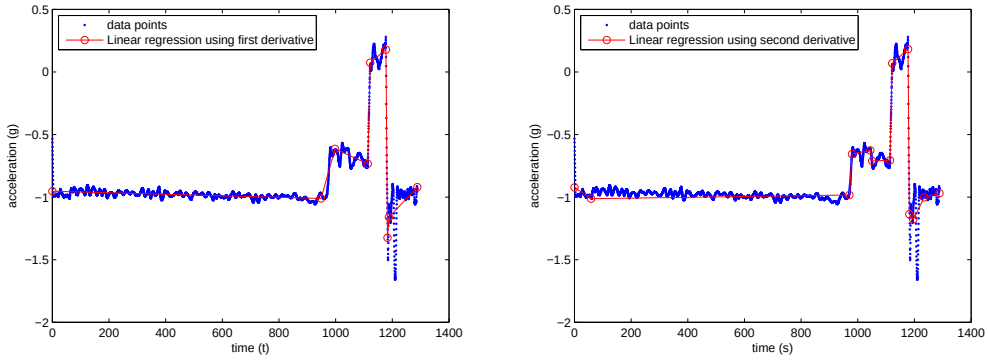

Figure 12: The linear regression found by using the first derivative (left) and second derivative (right) to find breakpoints.

1. Apply median filter
2. Find points using a threshold on either the first or second derivative
3. Apply postprocessing algorithms

Both the first and second derivative can give good results as can be seen in Figure 12. The parameters have been chosen by trial and error and should probably be determined by using several data sets.

We also applied these algorithms (with different parameters) to the the jump only. This could result in some insight in the movement of the skydiver as his drag will be different depending on his position. But it appears that it is very hard to model this properly, see Figure 13. We also tried to apply splines to the same breakpoints, but this result was worse. It looks like the linear regression works relatively well for the general acceleration, but not good for the jump only. Since the jump is the most important part, we want to use a more precise method.

We can check if the acceleration and height data are correlated. They should be correlated by the velocity for example. We can check this by approximating it by using the height or the acceleration as can be seen in Figure 14. Also the linear regression was used to find a better approximation of the velocity derived from the acceleration. Note that a constant cannot be derived and therefore the functions are not perfectly aligned, but the shape is the same. The velocity derived from the height has a larger error, but behaves more or less the same.

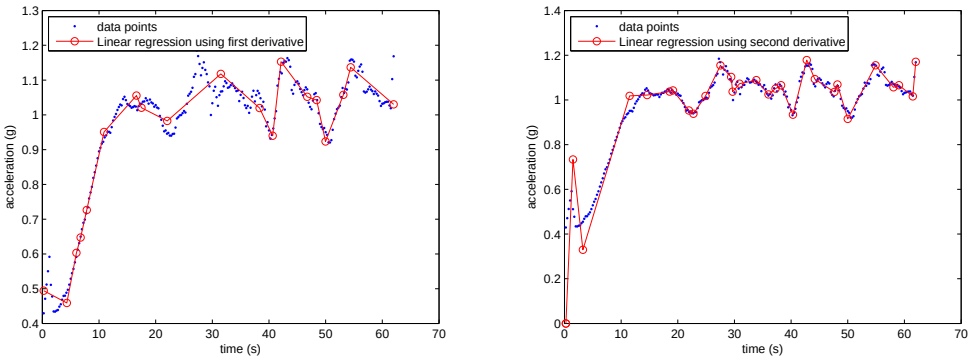

Figure 13: The linear regression found by using the first derivative (left) and second derivative (right) to find breakpoints.

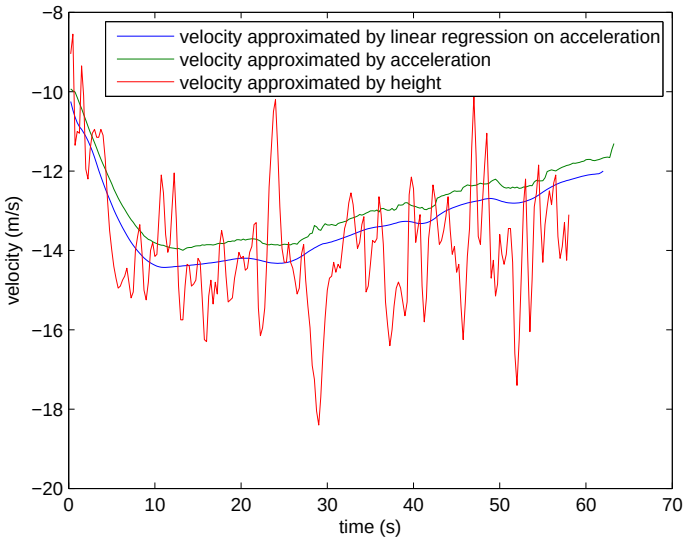

Figure 14: Velocity approximated by using the height, acceleration and linear regression of acceleration for the jump