

Camera-Based Localization and Stabilization of a Flying Drone

Jan Škoda and Roman Barták

Faculty of Mathematics and Physics

Charles University in Prague

skoda@jskoda.cz, bartak@ktiml.mff.cuni.cz

Abstract

This paper describes implementation of the system controlling a flying drone to stabilize and hold the drone still regardless of external influences and inaccuracy of sensors. This task is achieved by utilizing visual monocular SLAM (Simultaneous Localization and Mapping) – tracking recognizable points in the camera image while maintaining a 3D map of those points. The output location is afterwards combined using the Kalman filter with odometry data to predict future location using drone’s dynamics model. The resulting location is used afterwards for reactive control of drone’s flight.

Self-regulation of systems is a long-time studied subject with many techniques developed especially in the area of control theory. When we know the current state of the system then it is possible to use one of existing controllers to reach (and keep) the desired state of the system. The problem here is not finding the path between the states, which is a topic of planning and it is easy in this case, but rather controlling the real system to reach the desired state as soon as possible without overshooting and oscillating.

In this paper we address the problem of keeping a flying drone still even under external disturbances. Our ambition is using only the sensors available on the drone to estimate the current state, location in our case, of the drone, which is the most challenging part of the stabilization problem. In particular, we are working with AR.Drone belonging to the category of robotic toys, but still providing a reasonable set of sensors that makes AR.Drone a useful research tool too. Similarly to humans, the most informative sensor is a camera, which is also a key source of data for visual localization used in the proposed system. Due to limited computation power of the onboard processor, all processing is realized on a connected computer (mainstream laptop), which brings another challenge in time delay between observation and acting. In summary, we propose a system that does visual localization of a flying drone and uses information about drone’s location to keep the drone still.

Previous work on mobile robot localization was done in several fields. Wheeled vehicles often use some kind of relative localization based on odometry, but that is not very useful for flying drones. Absolute localization of UAVs using

external sensors was implemented using cameras (Krajník et al. 2013). Furthermore, down-looking cameras have been used to stabilize a UAV in (Chudoba et al. 2014), but the method encountered problems with insufficiently textured ground surfaces lacking distinguishable landmarks. Another approach is to utilize external beacons (Krejcha and Vechet 2012). The last two methods are very precise, but require external support, which limits their usage to prepared environments. For outdoor flight, GPS-based localization can be used. The AR.Drone 2 is compatible with a complete solution, *Flight Recorder* device, which integrates a GPS module. Finally, various SLAM systems using only onboard sensors were implemented utilizing ranging sensors (Achtelik et al. 2009) or camera (Engel, Sturm, and Cremers 2012). The system described in (Engel, Sturm, and Cremers 2012) is very similar to ours as it implements visual SLAM and stabilization for AR.Drone 1. This system was the major inspiration for our work.

The paper is organized as follows. First, we briefly describe the robotic platform used, AR.Drone 2 by Parrot, as its hardware specification influences decisions done in this project. Then we overview the proposed approach and give some details about the used techniques for visual localization and mapping. After that we describe how an extended Kalman filter is used to tackle the problem with time lag and what type of controller we use. The paper is concluded by a summary of experimental evaluation of the implemented system. We show how the system behaves in different environments.

AR.Drone Platform

AR.Drone 2.0 by Parrot Inc. is a robotic platform originally intended as a WiFi-controlled flying toy for capturing videos and playing augmented-reality games. Drone movement is controlled by adjusting speed of four rotors (Figure 1), which is done by the drone’s firmware according to higher-level commands (see below). The main advantages of the platform are its very low price, robustness to crashes, and the wide variety of onboard sensors.

The AR.Drone is equipped with two cameras, one facing forward and one downward. The bottom camera is used by the firmware to estimate the vertical speed. That is however inaccurate and works only above well-textured surfaces. The forward HD camera is used for recording video

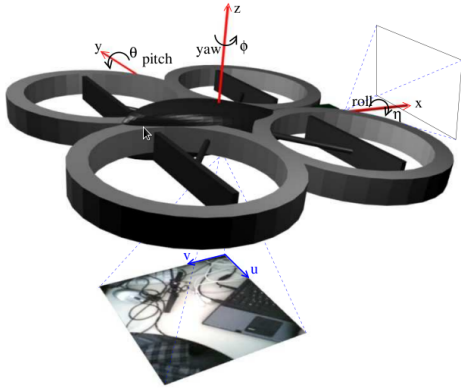


Figure 1: AR.Drone and its coordinate system and angles. (Krajník et al. 2011)

on attached USB flash storage and/or streaming the video over a WiFi network. The video stream is unfortunately aggressively compressed and especially during movement heavily blurred. The drone is further equipped with a 3-axis gyroscope, a 3-axis accelerometer, and a magnetometer. Altitude is measured using an ultrasound sensor and a pressure sensor, which is used in higher altitudes out of the ultrasound sensor's range.

The drone contains a control board with a 1 GHz ARM Cortex processor running a minimalistic GNU/Linux system. It is technically possible to run own programs directly onboard, but because of the computing power required to process the video we use an external computer to control the drone remotely. The AR.Drone creates a WiFi access point with a DHCP server, so that the controlling device can easily connect and communicate using UDP connections. The flight is generally controlled just by sending pitch and roll angles and vertical and yaw speed. The commands are sent at 30 Hz and the drone's firmware then tries to reach and maintain given values until the next command arrives. Data from non-visual sensors, so-called *navdata*, are sent from the drone at 15-200 Hz depending on setting and contains especially roll and pitch angles, azimuth, altitude, and a speed vector in the drone centered coordinate system (Figure 1). The network latency of transmission of those commands and data is approximately 60 ms.

The video from the front camera is (in the default setting) downscaled to 640x360 px, encoded using H.264 with a maximum bitrate of 4 Mbps and streamed over UDP at 30 FPS. The latency between capturing the image and receiving it at the drone's board is about 120 ms.

Overview of the approach

This work utilizes SLAM (Simultaneous Localization and Mapping) techniques to localize the drone and stabilize it in a desired position. There are many other approaches to the localization problem such as using a GPS signal or artificial markers or transmitters distributed in the environment.

That would however limit the usage of the system to carefully prepared environments. Another attempt to evade the necessity of SLAM implementation would be to use only relative localization techniques such as optical flow tracking or acceleration-based speed estimation. Such techniques are however unable to eliminate drift and localization error grows over time, so the techniques are applicable just for a limited time.

For visual localization, we use a system based on the *PTAM* library (Klein and Murray 2007). The system receives a video frame at 30 Hz together with a pose prediction based on previous state estimation. It outputs the most-likely pose estimate of the drone relative to the starting position together with the precision specifier. That position is processed in an *Extended Kalman Filter (EKF)* (Welch and Bishop 1995) together with other measurements received in *navdata* such as speed estimate. When the visual tracking is considered lost, the EKF ignores the visual pose estimate and predicts the relative pose change from *navdata* only.

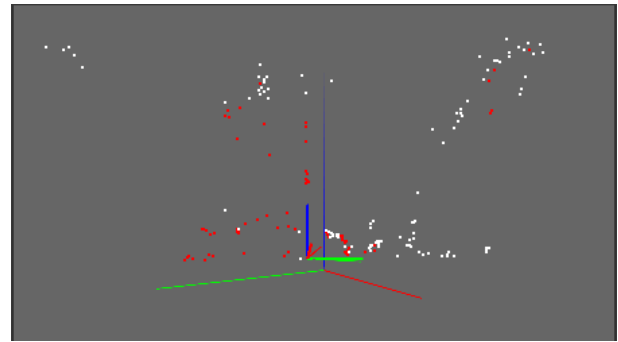


Figure 2: Visualization of the map and the drone's pose. Red landmarks are those currently observed.

EKF contains a probabilistic motion-model of the drone's flight dynamics and it is an important addition to the visual localization for several reasons. It combines the visual pose estimate with other measurements to increase the estimate precision and maintains it even when the visual system fails and no absolute pose is measured. Finally, EKF is able to accurately predict drone's movement for a short time, which is used to balance the long network latency. The control commands executed on the drone are based on almost 0.2 s old data. That would result in inaccurate motion and oscillation around the stabilization position. EKF solves that problem by providing a 0.2 s prediction of the pose to the control system.

The usage of a single camera introduces several challenges for the SLAM system. It is possible to estimate the bearing of a point in a video frame with the knowledge of the camera model (focal length, distortion parameters), but the distance of the point can not be measured. That is a problem when we want to add an observed landmark to the map. For that we need more observations of the same landmark from different positions (Figure 3). That is a problem when the drone is stabilized, as the distance of positions (and the angle γ in the Figure 3) is small and the estimated distance is

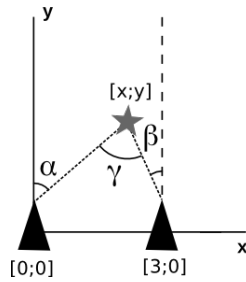


Figure 3: Localization of a landmark in 2D.

inaccurate. We have therefore decided, that the map will be prepared before the stabilization (but possibly after takeoff) as the localization quality strongly depends on the precision of the map.

The built-in camera is unable to provide directly the scale of distances in the picture compared to the real world. This scale is required for the control system to measure the distance to the desired position on which the approach speed depends. The scale can be estimated using other measurements of movement in the *navdata* (Engel, Sturm, and Cremers 2012), but in this work, we estimate the scale during initialization of the visual localization system, which is required for inserting the first landmarks into the map.

When the system knows the drone's and the desired poses, it uses PID controllers to reach and maintain the pose. One controller is utilized for each coordinate of the 3D position and for the azimuth.

Visual Localization and Mapping

To estimate a pose of the drone from the received video frames, our software uses SLAM system based on *Parallel Tracking and Mapping* method (Klein and Murray 2007) and this section provides a short overview of the method. PTAM was developed to track hand-held camera motion in unknown environment. The tracking and mapping are split into two tasks processed in separate threads, which can be run in parallel on a dual-core computer so that computationally expensive batch optimization techniques can be used for building the map. The resulting system is very robust and accurate compared to other state-of-the-art systems – in the cited paper, it was successfully compared to the widely used *EKF-SLAM*.

In order to localize the drone, the system maintains a map of landmarks observed in the environment (Figure 2). The map is not updated for every frame, only for certain *keyframes*. Keyframe composes of a video frame, a set of keypoints detected by the FAST corner detector (Trajković and Hedley 1998), and a pose estimation, which can be later updated in order to increase the precision of the pose and therefore even the precision of the associated keypoints locations. The structure of the map is illustrated in Figure 4.

The map has to be initialized before the localization. This is done by inserting first two keyframes which define the origin of the coordinate system and its scale to the real world. The points observed in those keyframes are then used as the

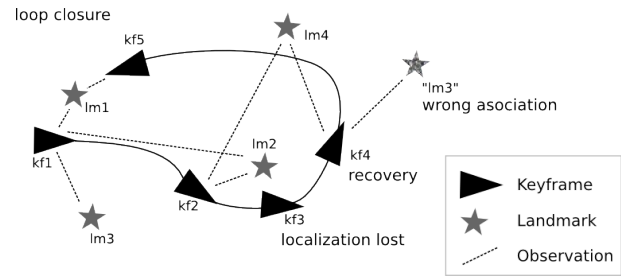


Figure 4: Graph representation of the internal map.

first landmarks in the map and their positions are calculated using the five point algorithm (Stewenius, Engels, and Nistér 2006). This procedure requires the user to press a keyboard button to insert the first keyframe into the map, move the drone 10 cm to the right and press the button again to insert the second keyframe. The distance must be known by the system and can be arbitrary, but too small translation compared to scene depth would result in worse precision (small angle γ in Figure 3) of the triangulation. The scale of the map could be estimated using the accelerometer as well. Unfortunately, the AR Drone 2 does not provide the acceleration measurements before takeoff.

As mentioned above, landmarks are added to the map only when a keyframe is inserted. More specifically, a landmark can be localized only after its second observation, when the landmark's location can be measured using triangulation (Figure 3). The two keypoints of observation of a single landmark are associated using epipolar search (Faugeras 1993) and zero-mean SSD (Nickels and Hutchinson 2002) for their pixel patches. Notice that as the computed location of the landmark is relative to the location of the drone, the error of the landmark's location is affected by the error of the drone's location. As the precision of the map is critical for further localization, we will later describe the means of improving the precision using subsequent observations.

Camera Pose Estimation

Having the map, we can compare it with landmarks observed in every frame to localize the drone. In this section we will briefly describe how this is done.

Assume that we have a calibrated pin-hole camera projection model *CamProj*:

$$\begin{pmatrix} u_i \\ v_i \end{pmatrix} = \text{CamProj} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad (1)$$

Where x, y, z are the coordinates of a landmark relative to the current camera pose and u_i, v_i are the (pixel) coordinates of the landmark projection into the image plane of the camera. Let *CameraPoseTrans*(μ, p_i) denote the location of the landmark p_i relatively to the camera pose μ . We can use the defined projection to express the reprojection error vector e_j of the landmark with coordinate vector p_j (relative to the map origin) which was observed at u_j, v_j . Reprojection error is the difference between where the landmark p_j

should be observed according to the map, if the drone's pose is μ , and where it was observed using the camera.

$$e_j = \begin{pmatrix} u_j \\ v_j \end{pmatrix} - \text{CamProj}(\text{CameraPoseTrans}(\mu, p_j)) \quad (2)$$

In the correct pose of the drone, the reprojection errors should be very small. Therefore we can use e_j for finding the most-likely camera pose μ' :

$$\mu' = \underset{\mu}{\operatorname{argmin}} \sum_{j \in S} \text{Obj} \left(\frac{e_j}{\sigma_j}, \sigma_T \right) \quad (3)$$

where S denotes the set of landmark observations, $\text{Obj}(\cdot, \sigma_T)$ is the Tukey biweight objective function (Hampel, Ronchetti, and Rousseeuw 1986), and σ_T is a robust estimate of the distribution's standard deviation.

Mapping

Mapping is a process of adding newly observed landmarks into the map and updating the pose of known landmarks after further observations in order to improve the precision of their location. All mapping operations, which can be computationally expensive, are done in a separate thread.

We have already outlined the process of keyframe addition, in which the landmarks are added to the map. When the mapping thread doesn't work on that, the system use the spare time to improve the accuracy of the map. The position of a landmark is initially computed from its first two observations. We can improve that by minimizing the reprojection error of the landmark's location for all observations and landmarks.

Assume that we have N keyframes $\{1, \dots, N\}$. In each of them, we observed a landmark set S_i , which is a subset of a set $\{1, \dots, M\}$ of all M landmarks. We will denote the j th landmark observed in some keyframe i with the subscript j^i . μ_i is the pose of a keyframe i and p_j is the location of a landmark j . Bundle adjustment is then used to update the poses of keyframes and the locations of landmarks (in a similar way as in the equation 3):

$$\{\{\mu_2, \dots, \mu_N\}, \{p'_1, \dots, p'_M\}\} = \underset{\{\mu\}, \{p\}}{\operatorname{argmin}} \sum_{i=1}^N \sum_{j \in S} \text{Obj} \left(\frac{e_{ji}}{\sigma_{ji}}, \sigma_T \right) \quad (4)$$

Note that the pose of the first keyframe is fixed in the origin of the map, hence μ_2 .

Extended Kalman Filter

We employ an Extended Kalman Filter (*EKF*) (Welch and Bishop 1995) for state-from-measurements estimation. Its goals are noise filtering, processing multiple measurements of a single variable, and prediction of the state of the system in the near future. The extended version of KF is necessary due to the nonlinear nature of the drone's flight dynamics.

EKF stores the state as a (multivariate) normal distribution of X represented by its mean and covariance matrix.

Similarly, measurements are perceived as a normal distribution of Z with the mean value equal to the received measurement. Its covariance matrix is usually fixed and represents the precision of sensors. Finally, EKF receives a control vector, which describes the command sent to the drone. Relation between two subsequent states and the control vector u is defined by a *process model* $P(X_k|X_{k-1}, u_{k-1})$, relation between state and measurement is defined by a *measurement model* $P(Z_k|X_k)$. We will further denote the means of the state and the measurement at a time k as x_k and z_k . Note that the measurement model determines measurements from states to compare it with received measurements and not vice versa.

The major task of an EKF utilization is to implement the process and measurement models. Due to space limit we will not describe the whole implementation, especially the motion model, but only the interface of the filter and the main part of the measurement model. The interface between the EKF and the control system is composed mostly of the vectors x_k , z_k and u_k :

- $x_k = (x, y, z, v_x, v_y, v_z, \phi, \theta, \psi, d\psi)$ – 3D coordinates relative to map origin, 3D speed vector in the same system, roll, pitch, yaw and yaw rate
- $u_k = (\bar{\phi}, \bar{\theta}, \bar{\psi}, \bar{v}_z)$ – desired roll, pitch, yaw and yaw rate as sent to the drone
- $z_k = (v'_x, v'_y, v'_z, \phi, \theta, \psi, x, y, z)$ – measured speed in 3D coordinates relative to the drone (Figure 1), roll, pitch, yaw and the drone's coordinates in 3D from the visual localization system

The measurement model is used to *correct* the filter's prediction of the process state x_k according to the obtained measurement. The main part of the model is a function $z_k = g(x_k)$, which is used to compute the expected measurement to be compared with the measurement obtained from the drone.

$$\begin{pmatrix} v'_x \\ v'_y \\ v'_z \\ \phi \\ \theta \\ \psi \\ x \\ y \\ z \end{pmatrix} = g(x_k) = \begin{pmatrix} v_x \cos \psi - v_y \sin \psi \\ v_x \sin \psi + v_y \cos \psi \\ v_z \\ \phi \\ \theta \\ \psi \\ x \\ y \\ z \end{pmatrix} \quad (5)$$

Together with the function g , the measurement model contains a covariance matrix, which specifies the precision of sensors. When the visual localization system fails for a moment, the variances of it's output, location (x, y, z) , are increased, so that the filter practically ignores the measurements (x, y, z) and updates the pose of the drone according to the process model and the other measurements.

Drone Control

The control system receives the most-likely state prediction x_t , computes the control command from x_t and sends it to the drone for execution. The time t is the time of receiving

sensor measurements used to estimate x_t plus the expected latency after which the command will be executed on the drone. This way, the drone will react to its current pose and not to some older one.

The control command is obtained using four independent PID (proportional-integral-derivative) controllers for each degree of freedom: x, y, z, yaw . Let $e(t)$ denote the error of the controlled variable at time t . Then the output $out(t)$ of the PID controller is calculated according to the following classical formula:

$$out(t) = P \cdot e(t) + I \cdot \int_0^t e(t)dt + D \cdot \frac{de(t)}{dt} \quad (6)$$

where P, I and D are parameters (weights) of the controller which have to be tuned. They describe the reaction of the controller to the error (P), the integrated error (I), and the speed of change of the error (D). Note that after initial testing of the system, we have set the I parameter to zero in order to prevent the wind-up effect and overshooting.

From each of the four controllers we obtain a desired change of controlled variables: x_d, y_d, z_d and yaw_d . As the coordinates are relative to the map origin, we have to rotate x_d, y_d to the drone-centric coordinate system (Figure 1). Then we construct the control command u_t – we use x_d, y_d as the two tilt angles of the drone, z_d as the vertical speed and yaw_d as the rotational speed. Therefore $u_t = (x_d, y_d, yaw_d, z_d)$.

Evaluation

The performance and robustness of the system was experimentally evaluated by examination of the ability of the system to stabilize the drone in a given position. A series of measurements was made in various environments. As we unfortunately did not have any device capable of recording the true location of the drone (ground truth), we had to record and measure the results by hand. The measurements were performed according to this scheme:

1. The visual localization system is initialized.
2. Several keyframes (around five) are inserted manually.
3. The gyroscope is calibrated.
4. We manually fly with the drone to a desired position and enable the stabilization. The orthogonal projection of the drone’s location to the floor is marked on the floor. We used a pendulum to do that.
5. We push the drone approximately 20 cm aside.
6. After 20 s, we mark the drone’s location on the floor again and measure the distance, which is stated in the following tables as the *Measured error*.

Note that we didn’t measure the yaw or the altitude. It would only make the measurement longer, less precise and would not bring any new information, as the precision of the yaw and the altitude will be similar (or better thanks to the altimeter and the compass) than the precision of the x, y coordinates.

The measurement was done in several different environments with distinct number of detected landmarks, both in interiors and exteriors. The following tables summarize the results of the experiments:

Name	<i>College room</i>
Environment	Visually rich environment, small interior
Keypoints	approx. 200
Measured error	7 cm
Notes	The visual localization was lost between the initialization and takeoff as the drone laid on the floor was not able to observe the scene. However, after take-off the localization was immediately restored. Error fluctuated, but did not show a trend to grow in time.

Name	<i>House frontage</i>
Environment	Visually poor environment, enough light, light wind
Keypoints	approx. 100
Measured error	10 cm
Notes	The system maintained the localization. It was however almost unable to find any keypoint on the wall of the house.

Name	<i>Gymnasium</i>
Environment	Big room, artificial light
Keypoints	approx. 50
Measured error	3 cm
Notes	-

Name	<i>Bare wall and radiator</i>
Environment	Visually very poor environment, repeated patterns
Keypoints	approx. 15
Measured error	-
Notes	We managed to initialize the localization system, but the drone held in the desired position just for a few seconds and the measurement had to be aborted. Some landmarks created on the surface of the radiator were often observed in another parts of the radiator, which disrupted the localization.

The video demonstrating the system and showing its user interface can be found at <http://vimeo.com/102528129>.

Conclusion

The goal of the work is to implement a system able to stabilize the drone using localization techniques. The flying drone has to hold still regardless of external influences, inaccuracy of sensors, and the latency of control. As we wanted the stabilization to work accurately for longer periods of time, we had to avoid the effect of accumulated error typical for relative localization. Therefore we decided to implement a visual SLAM system.

As the used AR.Drone has no stereo-vision camera, the system has to be able to estimate the distances of observed objects from multiple observations from different locations. That is complicated by the fact, that the goal of the system is to hold at one particular location, so we have to prepare a localization map before activation of the stabilization. The method also assumes that the environment is mostly static and contains detectable visual landmarks (e.g. a room containing only plain walls is problematic).

The robustness and precision of our method was evaluated by conducting an experiment consisting of several measurements in various environments. In the experiment we showed, that our system is able to stabilize the drone surprisingly well despite the poor quality of the video, which is generated by the chosen low-cost platform.

Acknowledgements

Research is supported by the Czech Science Foundation under the project P103-15-19877S.

References

- Achtelik, M.; Bachrach, A.; He, R.; Prentice, S.; and Roy, N. 2009. Stereo vision and laser odometry for autonomous helicopters in gps-denied indoor environments. In *SPIE Defense, Security, and Sensing*, volume 7332. International Society for Optics and Photonics.
- Chudoba, J.; Saska, M.; Baca, T.; and Přeučil, L. 2014. Localization and stabilization of micro aerial vehicles based on visual features tracking. In *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, 611–616.
- Engel, J.; Sturm, J.; and Cremers, D. 2012. Camera-based navigation of a low-cost quadcopter. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 2815–2821. IEEE.
- Faugeras, O. 1993. *Three-dimensional computer vision: a geometric viewpoint*. MIT press.
- Hampel, F.; Ronchetti, E.; and Rousseeuw, P. 1986. Robust statistics: the approach based on influence functions. *Wiley series in probability and mathematical statistics*.
- Klein, G., and Murray, D. 2007. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, 225–234. IEEE.
- Krajník, T.; Vonásek, V.; Fišer, D.; and Faigl, J. 2011. Ar-drone as a platform for robotic research and education. In *Research and Education in Robotics-EUROBOT 2011*. Springer. 172–186.

Krajník, T.; Nitsche, M.; Faigl, J.; Duckett, T.; Mejail, M.; and Přeučil, L. 2013. External localization system for mobile robotics. In *Proceedings of the International Conference on Advanced Robotics, Montevideo*.

Krejsa, J., and Vechet, S. 2012. Infrared beacons based localization of mobile robot. *Elektronika ir Elektrotechnika* 117(1):17–22.

Nickels, K., and Hutchinson, S. 2002. Estimating uncertainty in ssd-based feature tracking. *Image and vision computing* 20(1):47–58.

Stewenius, H.; Engels, C.; and Nistér, D. 2006. Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing* 60(4):284–294.

Trajković, M., and Hedley, M. 1998. Fast corner detection. *Image and vision computing* 16(2):75–87.

Welch, G., and Bishop, G. 1995. An introduction to the kalman filter. In *Annual Conference on Computer Graphics and Interactive Techniques*, 12–17.