Nonlinear Control Framework for Gimbal and Multirotor in Target Tracking

Jae Hun Lee

March 16, 2018

(a) My little family

(b) Woojoo and Suho

(c) Family in Korea

(d) Hospital view

Group objective: To track multiple targets robustly using R-RANSAC.

My objective: To ensure targets are not lost in the camera field of view. Image-based visual servoing (IBVS) provides a good framework.

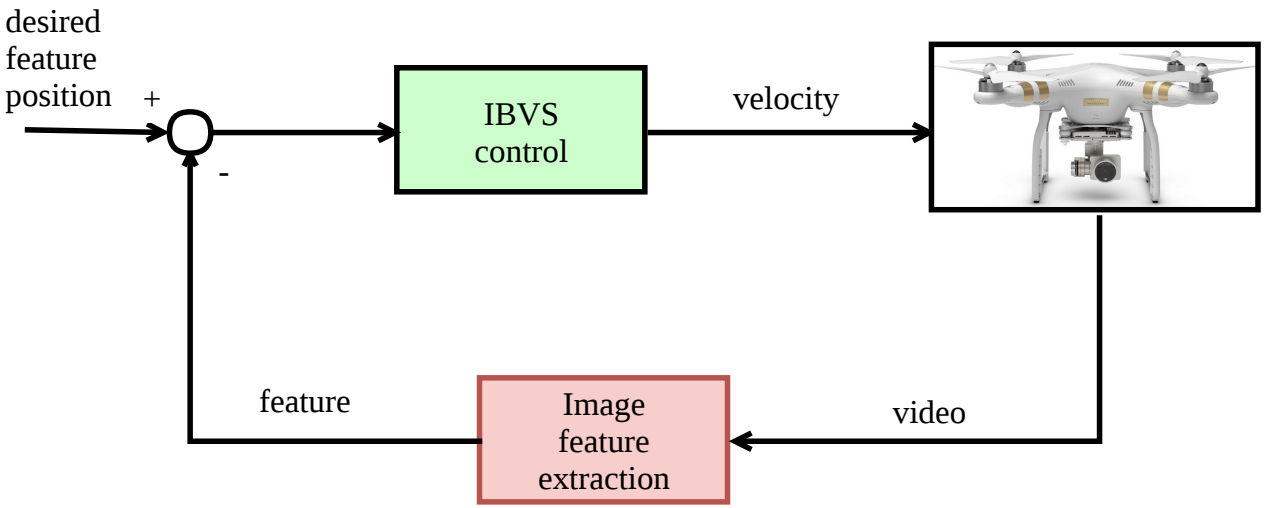


Figure: Image-based visual servoing block diagram

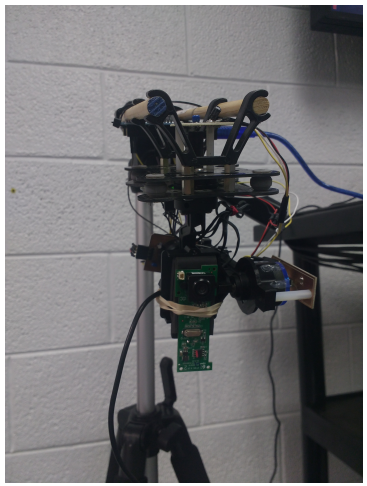
My question: How can I eliminate the needs for the depth term for visual servoing?

My contributions

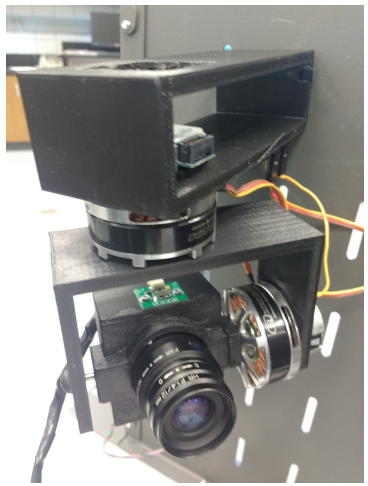
1. presenting three gimbal control algorithms including newly developed adaptive depth gimbal control.
2. integrating the system for multirotor autonomous target following using R-RANSAC tracker and demonstrating the hardware results.
3. developing the unit vector visual servoing framework for UAV control in target tracking.

Introduction

(a) Ideal



(b) First webcam version



(c) Custom uEYE camera gimbal

Figure: Gimbals

GImbal algorithms

1. Angle commanding gimbal control (UAV book)
2. Angular rate commanding gimbal control (Hurak and Rezac)
   - introduces image jacobian for gimbal control
3. Adaptive depth gimbal control (Me)
   - Model Reference Adaptive Control (MRAC) scheme eliminates the depth term in image jacobian

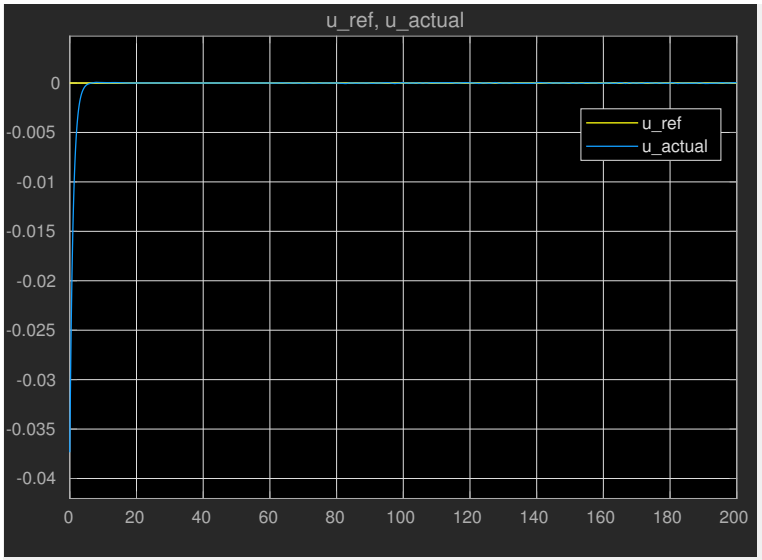
Image jacobian

$$\begin{bmatrix} \dot{u} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} -\frac{\lambda}{z} & 0 & \frac{u}{z} & \frac{uw}{\lambda} & -\frac{\lambda^2+u^2}{\lambda} & w \\ 0 & -\frac{\lambda}{z} & \frac{w}{z} & \frac{\lambda^2+w^2}{\lambda} & -\frac{uw}{\lambda} & -u \end{bmatrix} \begin{bmatrix} v_x^c \\ v_y^c \\ v_z^c \\ \omega_x^c \\ \omega_y^c \\ \omega_z^c \end{bmatrix}$$

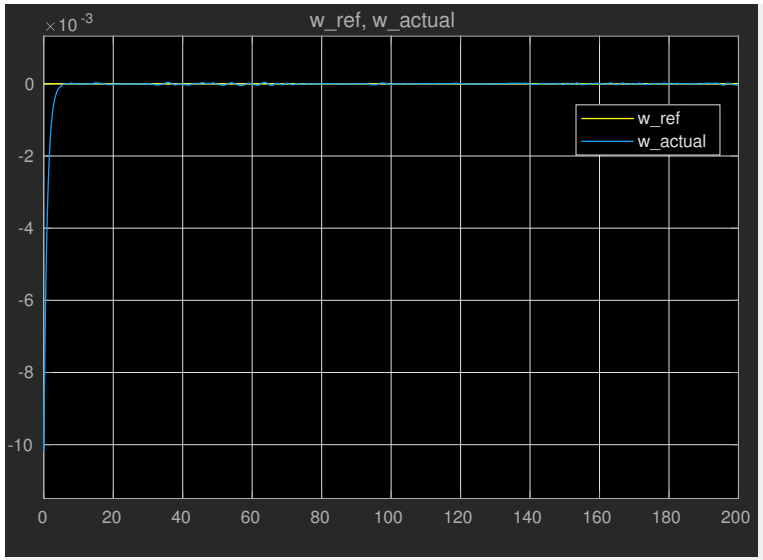
rearranged into

$$\dot{s} = \frac{1}{z} \begin{bmatrix} -\lambda v_x^c + u v_z^c \\ -\lambda v_y^c + w v_z^c \end{bmatrix} + \begin{bmatrix} \frac{uw}{\lambda} & -\frac{\lambda^2+u^2}{\lambda} & w \\ \frac{\lambda^2+w^2}{\lambda} & -\frac{uw}{\lambda} & -u \end{bmatrix} \begin{bmatrix} \omega_x^c \\ \omega_y^c \\ \omega_z^c \end{bmatrix}.$$

Simulation results - ADGC proof of concept



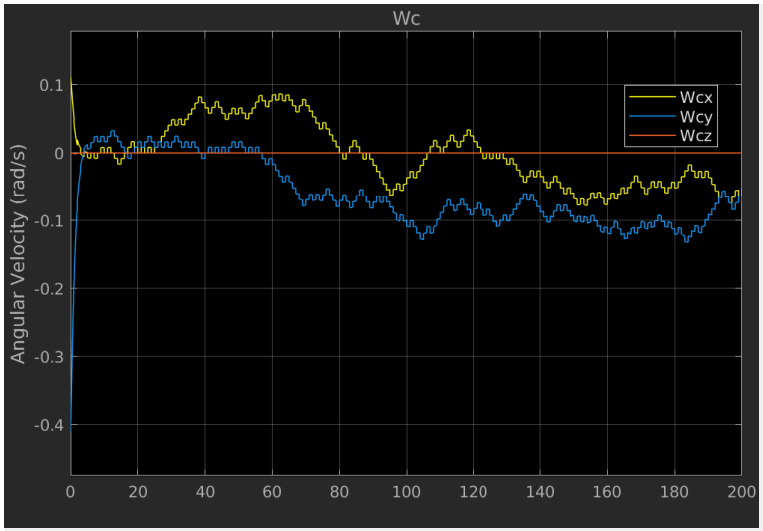
(a) Reference image coordinate u_{ref} and the system output u



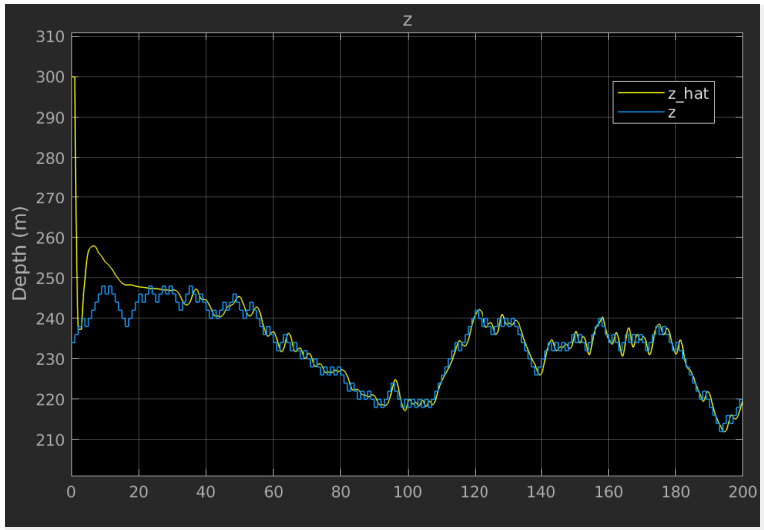
(b) Reference image coordinate w_{ref} and the system output w

Figure: The simulation result for the adaptive depth gimbal control. The system output u and w are converging to the reference model output u_{ref} and w_{ref}.

Simulation results - ADGC proof of concept



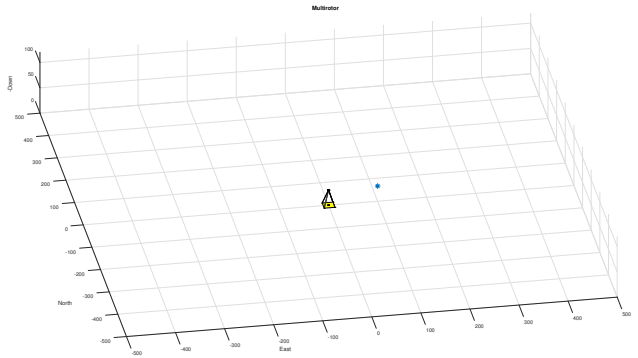
(a) Angular velocity commands



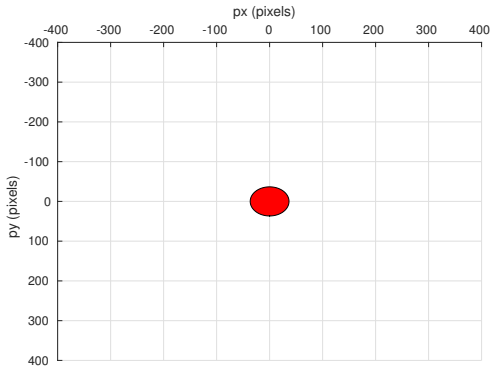
(b) Online depth estimation, $\hat{z}$

Figure: Angular velocity commands of the adaptive depth gimbal controller. Note that only two commands are used, since it is a pan-tilt gimbal. The depth z estimate using MRAC.

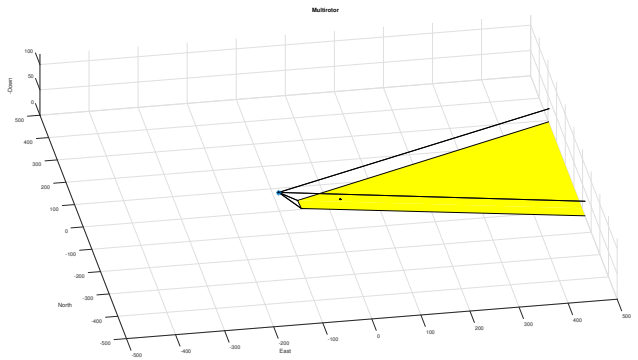
Simulation results - ADGC on multirotor



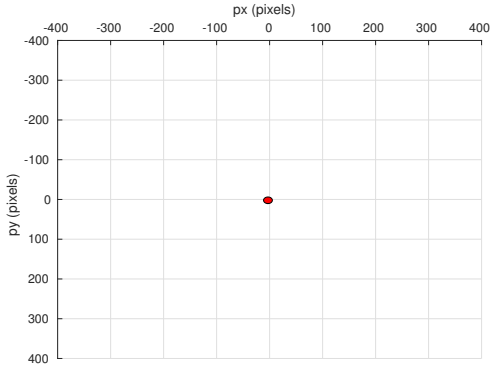
(a) Multirotor at $t = 0s$.



(b) Camera view at $t = 0s$



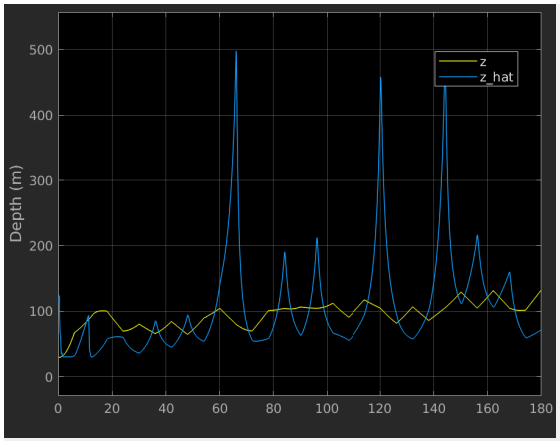
(c) Multirotor at $t = 180s$



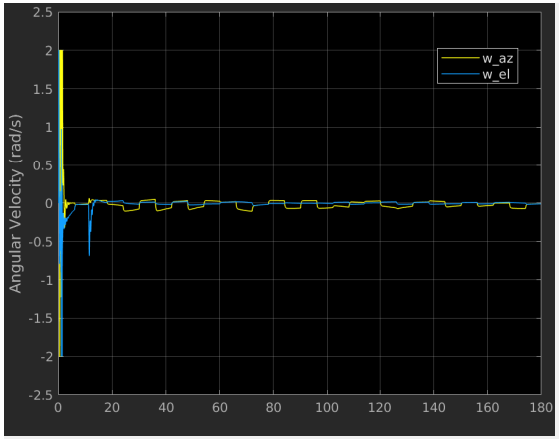
(d) Camera view at $t = 180s$

Figure: Multirotor simulation with camera view. The gimbal pointing objective is well achieved.

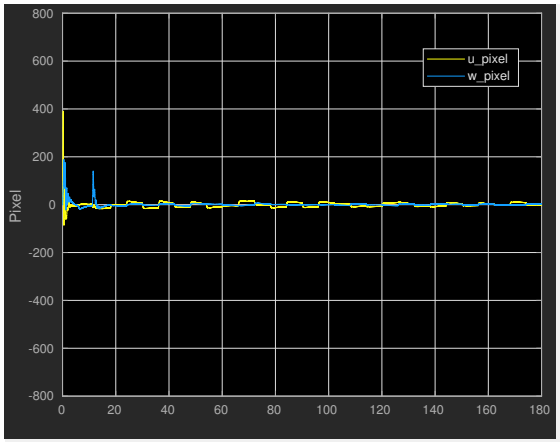
Simulation results - ADGC on multirotor



(a) Depth z and its estimate $\hat{z}$. Uncertain parameter converging to true value is not guaranteed.



(b) Angular velocity gimbal azimuth and elevation commands



(c) Pixel value u and w. They are maintained around the center of the image.

Figure: Uncertain parameter estimation, gimbal angular velocity commands from the controller, and where target lies in the image.

Hardware results - ADGC

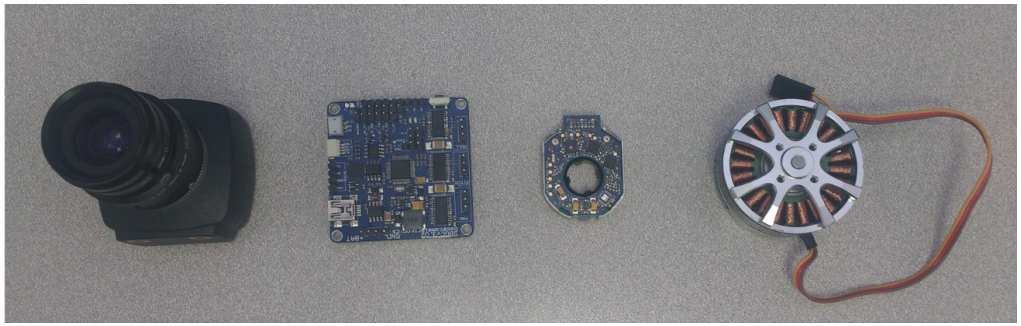


Figure: A custom pan-tilt camera gimbal

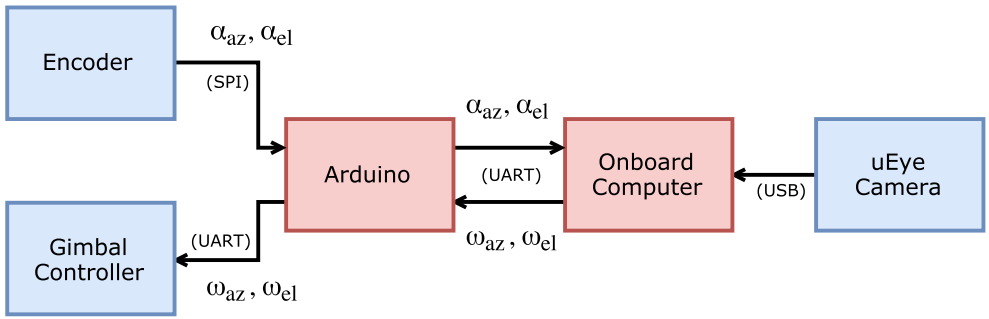


Figure: Custom gimbal block diagram

https://www.youtube.com/watch?v=VMfvQkhD9-o

ACGC

- Easy to implement
- Designed for static camera

AVCGC

- Designed for moving camera
- Requires the knowledge of the depth z

ADGC

- Works without knowing z
- Suitable for small UAV

System overview

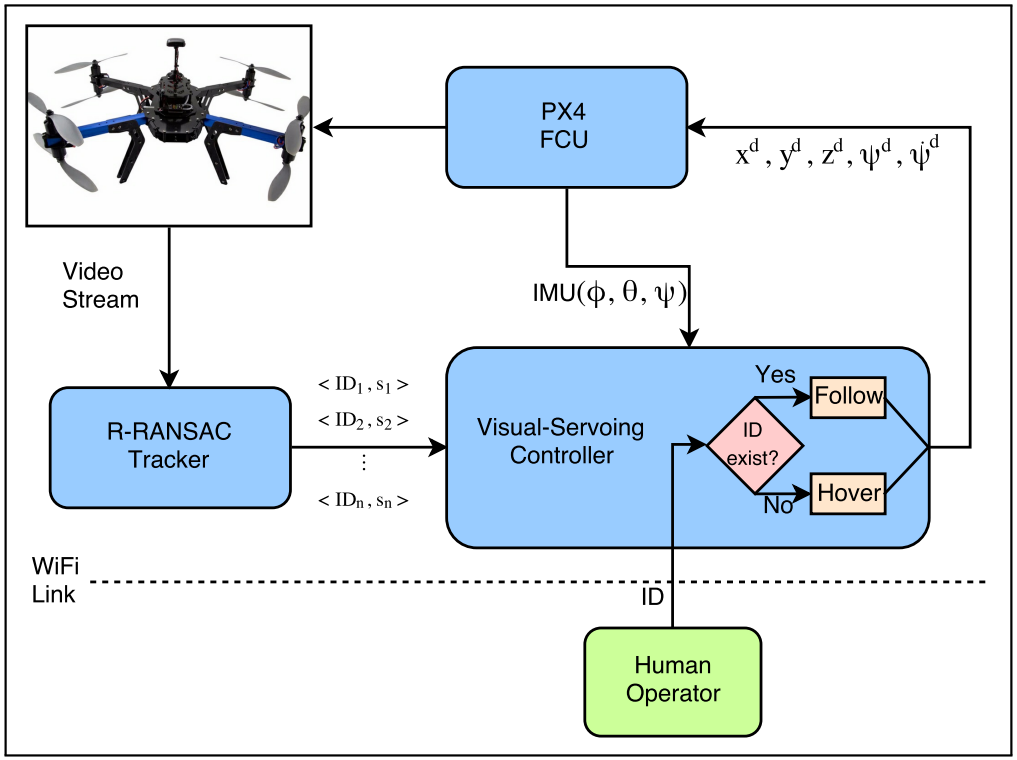


Figure: System Architecture. The R-RANSAC tracker produces a set of target ID numbers and corresponding pixel locations. The visual-servoing controller outputs the desired position, heading, and yaw rate based on the pixel location of the requested target.

Relatively simple controller using camera geometry

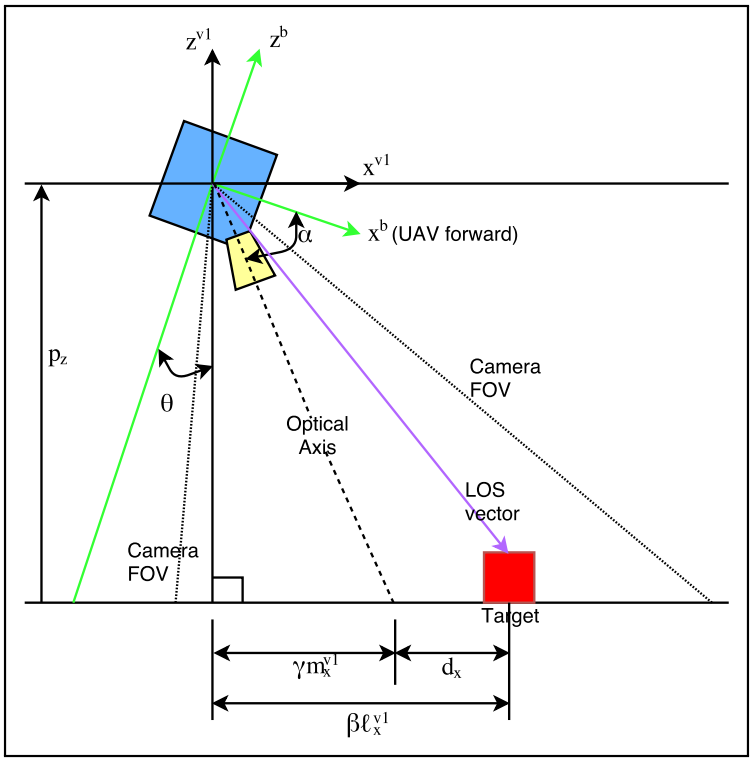


Figure: Side view of the multirotor.

Hardware results - camera view

(a) Track ID 51 initiated by R-RANSAC tracker (t=0s)

(b) The human operator has commanded the UAV to follow track ID 51 (t=13s)

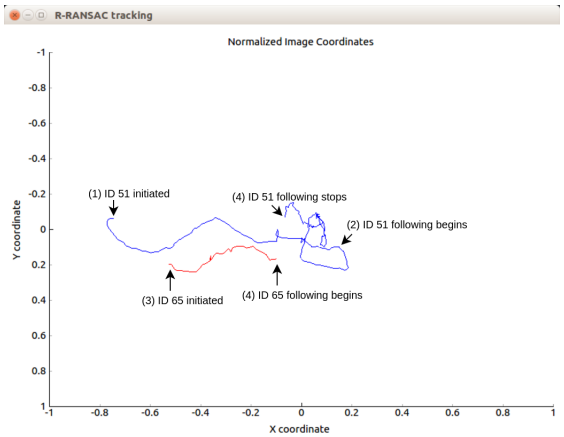
(c) Track ID 65 initiated by R-RANSAC tracker (t=27s)

(d) The human operator has commanded the UAV to follow track ID 65 (t=35s)

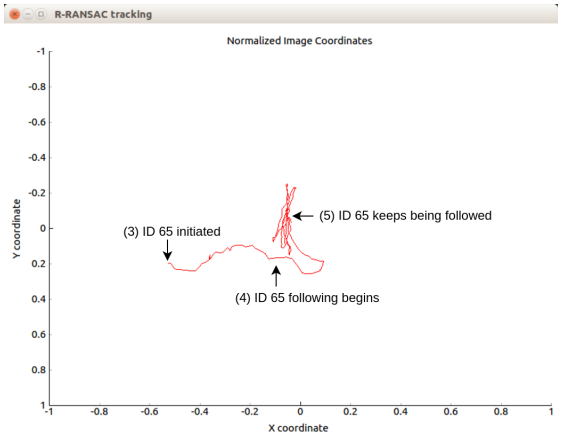
(e) A snapshot of the track ID 65 being followed (t=60s)

Figure: Camera view at various events

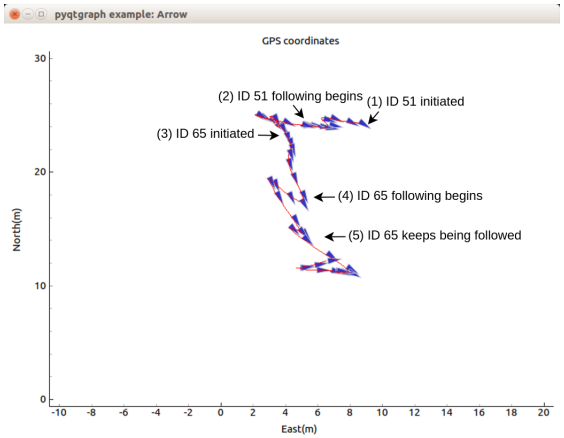
Hardware results - target footage in image and multirotor GPS footage



(a) Tracks movement in the normalized image plane. Each event (1)-(4) corresponds to camera view in 12a-12d respectively. Until the command to follow ID 65, the multirotor keeps the track ID 51 from leaving the camera view.



(b) The movement of track ID 65 in the normalized image plane. Each event (3)-(5) corresponds to camera view in 12c-12e respectively. The controller keeps the track ID 65 in the camera field of view after receiving the command to do so from the human operator.



(c) Multirotor GPS footage and heading corresponding to camera view in 12a-12e respectively.

Motivation

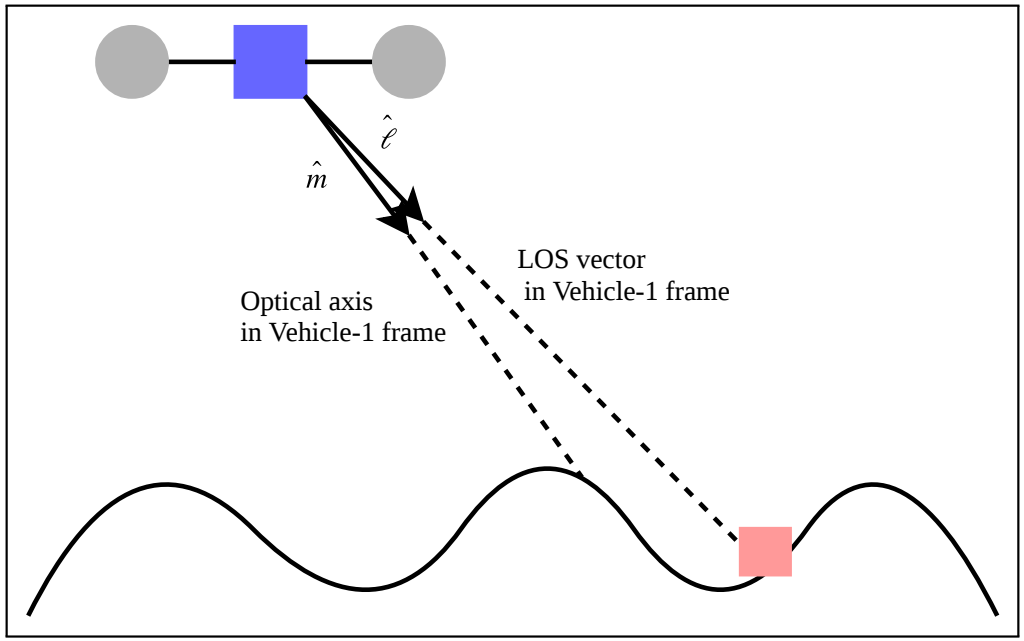


Figure: Non-flat-earth model example. The unit optical axis vector $\hat{m}$ and the unit line of sight vector $\hat{\ell}$ are key components of the controller presented in this chapter.

Control objective

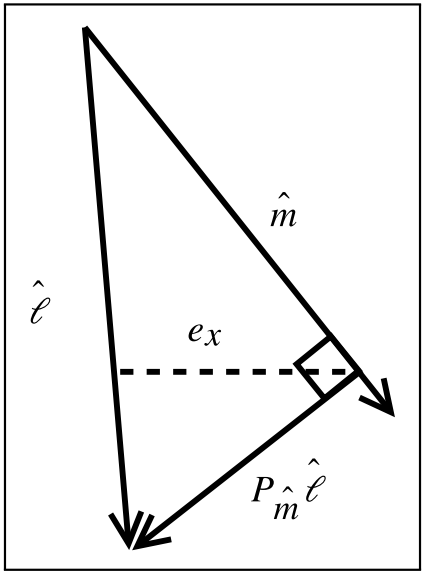


Figure: Projection onto the null space of the optical axis unit vector

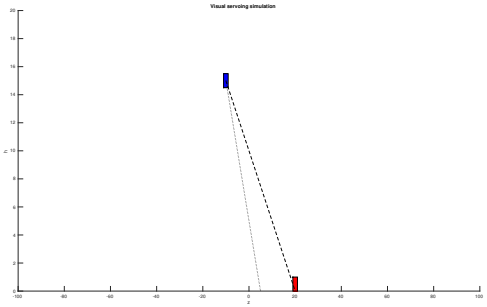
$$P_{\hat{m}} = (I - \hat{m}\hat{m}^\top)$$

$$\hat{e}_1 = [1 \quad 0 \quad 0]^\top$$

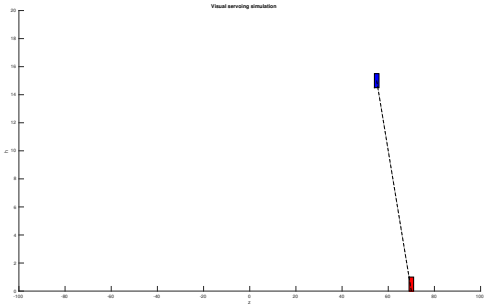
$$e_x = \hat{e}_1^\top P_{\hat{m}} \hat{\ell}$$

$$s = \dot{e}_x + k e_x$$

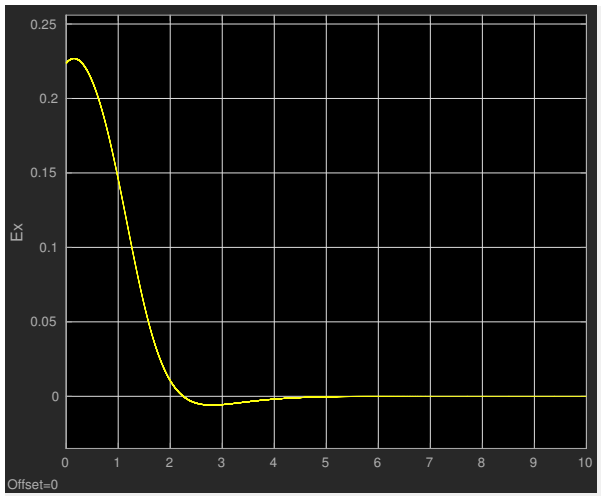
Simulation results - simple dynamics



(a) when $t = 0s$



(b) when $t = 10s$



(c) The horizontal error between the unit LOS vector and the unit optical axis vector converges to zero.

Figure: Simple UAV dynamics visual servoing Simulink simulation. The blue square is flying UAV at constant altitude and the red square is a target on the ground moving at $5m/s$. The initial UAV and target positions are [-10, 15] and [20, 0] respectively. Tuning parameters are set to $k = 1$, $\Gamma = I_3$ (identity matrix), and $\alpha = 1000$.

Backstepping controller derivation for multirotor dynamics

$$\ddot{p}_x = -\cos\phi \sin\theta \frac{F}{m}$$

$$\ddot{p}_x = -\frac{F_e}{m}\theta$$

$$\ddot{\theta} = \frac{1}{J_y}\tau_\theta$$

$$\vdots$$

$$\tau_\theta = \frac{J_y m}{\phi_1 F_e \hat{\beta}_1}(\xi_1 + \dot{\xi}_2 - \hat{\beta}_1\phi_1\frac{F_e}{m}\theta - \phi_1\frac{F_e}{m}\dot{\hat{\beta}}_1\dot{\theta} - k_3(\phi_1\frac{F_e}{m}\hat{\beta}_1\dot{\theta} - \xi_2))$$

Simulation system overview

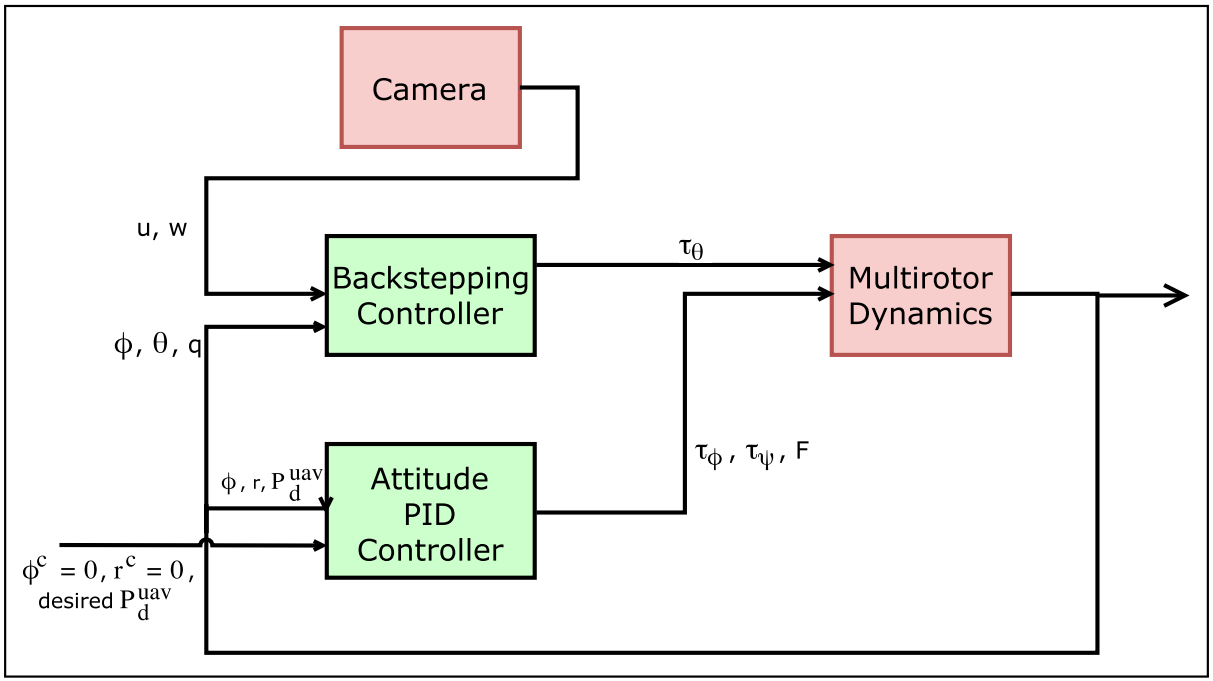
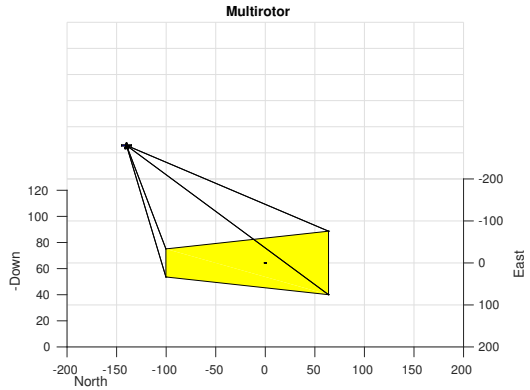
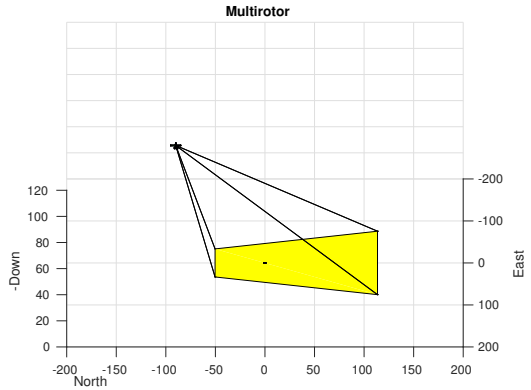


Figure: Control system diagram for the image-based backstepping controller. Note that the backstepping controller only requires the image coordinates of the target.

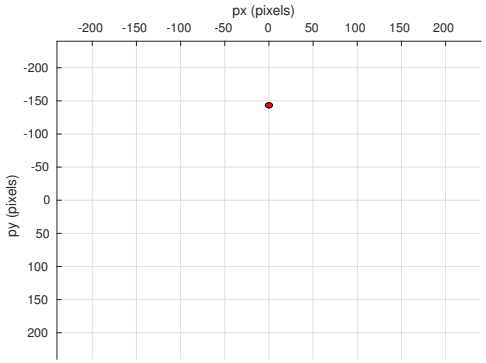
Simulation results - static target



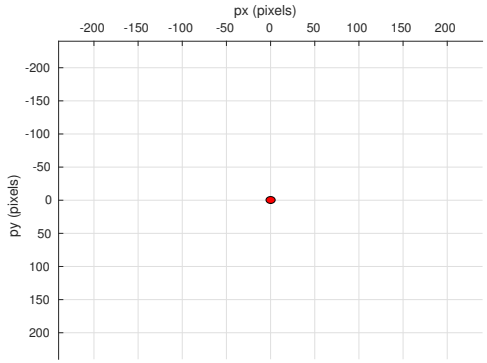
(a) Multirotor and ground target when $t = 0s$



(b) when $t = 60s$



(c) Camera view when $t = 0s$



(d) when $t = 60s$

Figure: Simulation result for the backstepping control using the normalized target pixel coordinates. The ground target is static ($0m/s$).

Simulation results - static target

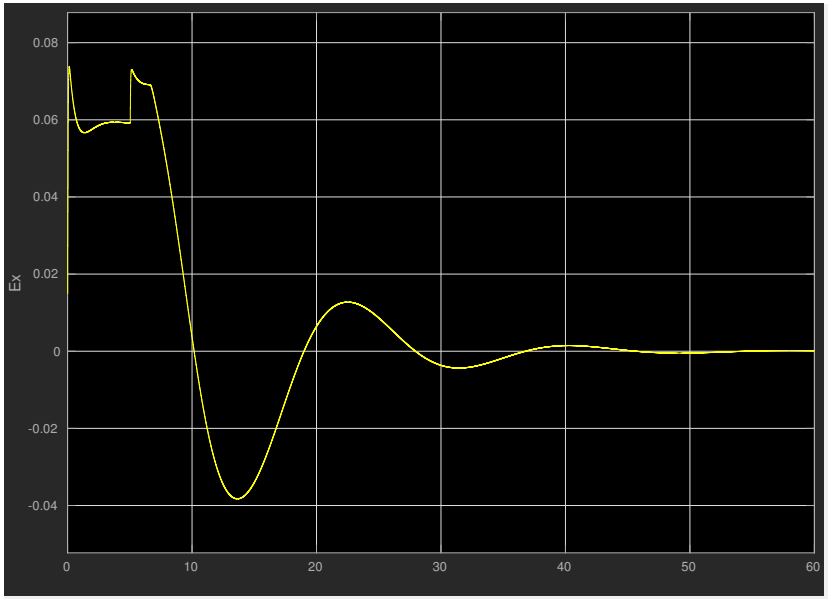
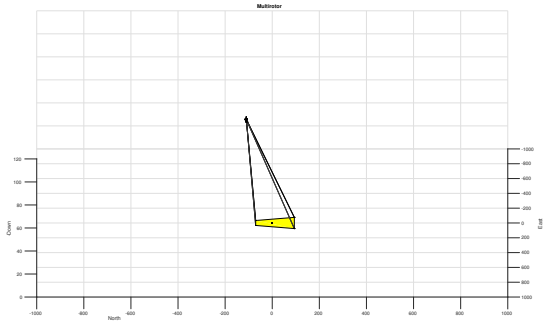
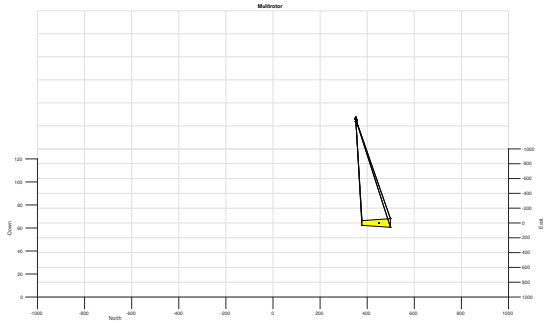


Figure: The horizontal error (e_x) between the normalized target pixel coordinates and the unit optical axis vector both in the vehicle-1 frame converges to zero. Note that the value is low-pass filtered.

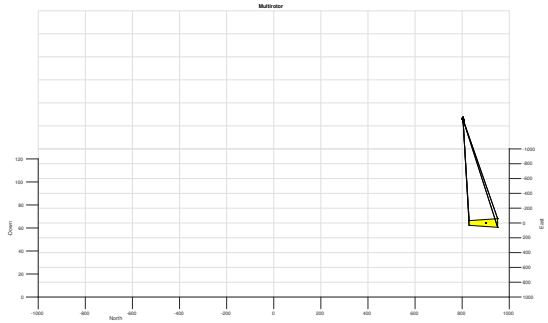
Simulation results - moving target



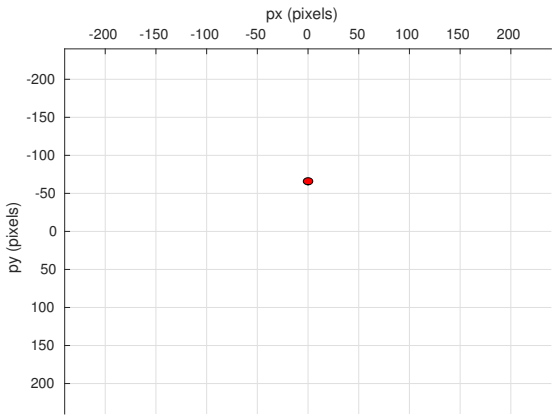
(a) when $t = 0s$



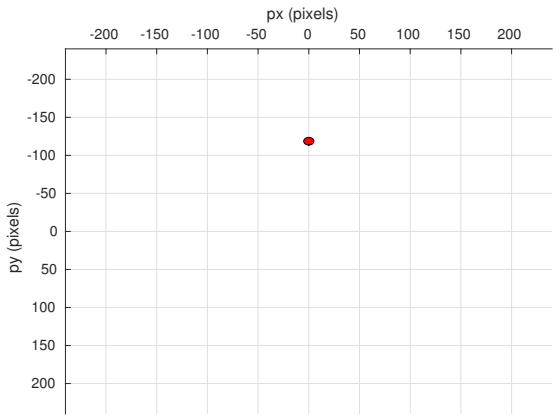
(b) when $t = 90s$



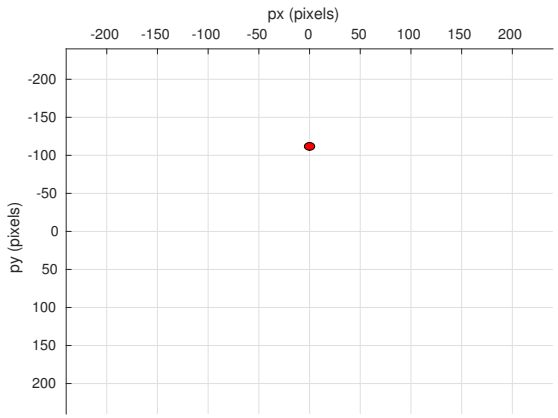
(c) when $t = 180s$



(d) when $t = 0s$



(e) when $t = 90s$



(f) when $t = 180s$

Figure: Simulation result for the backstepping control using the normalized target pixel coordinates. The ground target is moving at the speed of $5m/s$.

Simulation results - moving target

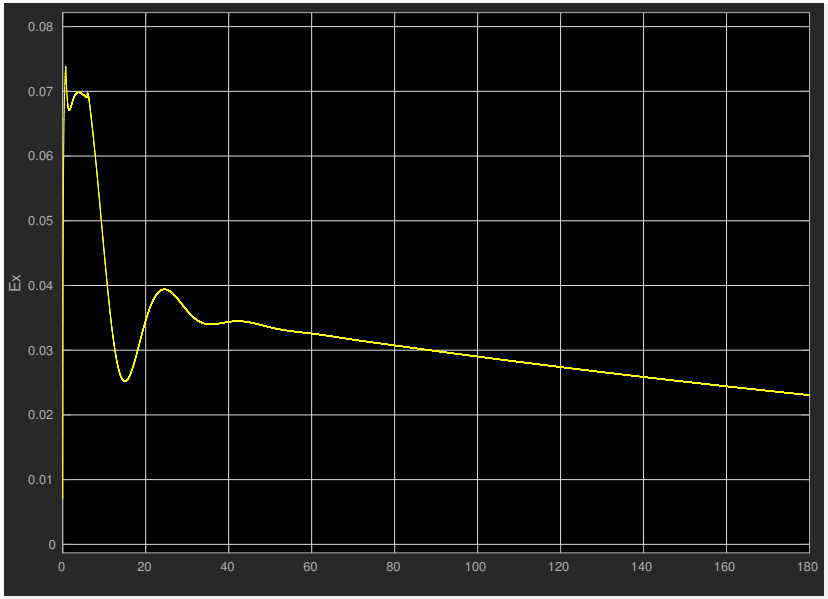


Figure: The horizontal error (e_x) between the normalized target pixel coordinates and the unit optical axis vector both in the vehicle-1 frame converges to zero. Note that the value is low-pass filtered.

Conclusion

- Adaptive depth gimbal control is useful when a depth sensor is not available and the camera is on a moving platform. Thus, it is suitable for small UAV tracking application.
- The first attempt to use the visual multiple target tracking using R-RANSAC tracker to close the UAV control loop in real-time. Theoretical contribution is not huge, the value sits in regard of the system integration and hardware result.
- A novel UAV control algorithm for target tracking, the unit vector UAV visual servoing, has been developed.

Future Work

- Implement the unit vector UAV visual servoing in Gazebo simulator. Test the control algorithm as well as the whole system with tracking algorithm. Tune the control parameter for more realistic multirotor and test with varying speed target. Come up with safety logic to ensure that the UAV is under control.
- Integrate the unit vector UAV visual servoing algorithm into the autonomous target following system in hardware. Test the system on non-flat ground.
- Design and build a new compact gimbal for small multirotor and test the adaptive depth gimbal control algorithm on a flying platform. Compare the result with other existing algorithms. (Optional)
- Implement the multiple target keeping in FOV algorithm from [**?**] and integrate into the autonomous target following system. Note that the algorithm from [**?**] is realistic for a multirotor equipped with at least two-axis gimbal. (Optional)