Ph.D. Thesis Defense:  
High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo  

Andrew Barry  
Robot Locomotion Group  
Massachusetts Institute of Technology
This thesis:
This thesis:

- 100% on-board perception
This thesis:

- 100% on-board perception
- 100% on-board computation
This thesis:

- 100% on-board perception
- 100% on-board computation
- No prior knowledge of the environment
Hard for the Right Reasons

Significant novelty required to fly around trees:
Hard for the Right Reasons

Significant novelty required to fly around trees:

1. Fast, lightweight sensing
Significant novelty required to fly around trees:

1. Fast, lightweight sensing
2. Fast control, integrated with sensing
Hard for the Right Reasons

Significant novelty required to fly around trees:

1. Fast, lightweight sensing
2. Fast control, integrated with sensing
3. Platform that can support (1) and (2)
Related works

Huge progress in the last 15 years:

- Larger UAVs
- Max takeoff weight 94 kg (145 times heavier than our aircraft)

Related works

Huge progress in the last 15 years:

- Larger UAVs \(^1,^2\)

---

\(^1\) Gavrilets et al., “Flight test and simulation results for an autonomous aerobatic helicopter”. 2002.

Related works

Huge progress in the last 15 years:

- Larger UAVs \(^1,^2\)

\(^1\) Gavrilets et al., “Flight test and simulation results for an autonomous aerobatic helicopter”. 2002.
Related works
Huge progress in the last 15 years:

- Larger UAVs $^{1,2}$

- Max takeoff weight 94 kg (145 times heavier than our aircraft)


Related works

- Micro aerial vehicles, or MAVs (under $\sim 5kg$)
- Highly aggressive trajectories in motion capture $3,4,5$

---


$5$ Barry et al., “Flying Between Obstacles with an Autonomous Knife-Edge Maneuver”. 2014.
Related works

- Flight through obstacles with a known map \(^6\)
- Environment not known until runtime \(^7\)


\(^7\)Majumdar and Tedrake, “Funnel Libraries for Robust Realtime Feedback Motion Planning”. 2016.
Related Work

Micro Aerial Vehicle (MAV) obstacle avoidance:
Related Work

Micro Aerial Vehicle (MAV) obstacle avoidance:

- 3D onboard
- 3D offboard
- Optic flow
- Prior map
- Motion capture

More Integrated Sensing

Speed (m/s)

- 0 2 4 6 8 10 12 14

This thesis
1: Beyeler, 2009
2: Mellinger, 2010
3: Bry, 2012
4: Barry, 2012
5: Richter, 2013
6: Ross, 2013
7: Shen, 2013
8: Dey, 2015
9: Oleynikova, 2015
Planning and Control

Good ideas exist:

- Differential flatness
- Nonlinear model predictive control (MPC)
- Trajectory libraries
- Time-varying linear quadratic regulators for stabilization (TVLQR)

Mellinger and Kumar, "Minimum snap trajectory generation and control for quadrotors". 2011.
Singh and Fuller, "Trajectory generation for a UAV in urban terrain, using nonlinear MPC". 2001.
Frazzoli, Dahleh, and Feron, "Robust hybrid control for autonomous vehicle motion planning". 2000.
Tedrake et al., "Learning to Fly like a Bird". 2009.
Planning and Control

Good ideas exist:

- Differential flatness $^8,^9$

Planning and Control

Good ideas exist:

- Differential flatness \(^8,^9\)
- Nonlinear model predictive control (MPC) \(^10\)

---

Planning and Control

Good ideas exist:

- Differential flatness \(^8,^9\)
- Nonlinear model predictive control (MPC) \(^10\)
- Trajectory libraries \(^11,^12\)

\(^12\) Stolle and Atkeson, “Policies based on trajectory libraries”. 2006.
Planning and Control

Good ideas exist:

- Differential flatness \(^8,^9\)
- Nonlinear model predictive control (MPC) \(^10\)
- Trajectory libraries \(^11,^12\)
- Time-varying linear quadratic regulators for stabilization (TVLQR) \(^13\)

---

\(^12\) Stolle and Atkeson, “Policies based on trajectory libraries”. 2006.
Sensing

Non-visual sensors:
Sensing

Non-visual sensors:

- LIDAR
Sensing

Non-visual sensors:

- LIDAR
  - localization in a map

Sensing

Non-visual sensors:

- LIDAR
  - localization in a map \(^{14}\)
- Kinect / active IR sensors

Sensing

Non-visual sensors:

- LIDAR
  - localization in a map \(^{14}\)
- Kinect / active IR sensors
  - indoor exploration \(^{15}\)


\(^{15}\) Michael et al., “Collaborative mapping of an earthquake-damaged building via ground and aerial robots”. 2012.
Vision

- Monocular vision
  - offboard depth estimation and control through a forest ¹⁶
- Embedded optical flow (optical mice sensors)
  - high rate, low resolution obstacle detection ¹⁷

Stereo Vision

On MAVs for a while now, generally too slow for fast flight. Fast stereo vision:

- GPU stereo
- FPGA stereo

Hrabar et al., "Combined optic-flow and stereo-based navigation of urban canyons for a UAV". 2005.
Byrne, Cosgrove, and Mehra, "Stereo based obstacle detection for an unmanned air vehicle". 2006.
Honegger et al., "Real-time velocity estimation based on optical flow and disparity matching". 2012.
Honegger, Oleynikova, and Pollefeys, "Real-time and Low Latency Embedded Computer Vision Hardware Based on a Combination of FPGA and Mobile CPU". 2014.
Stereo Vision

- On MAVs for a while now \(^{18,19}\)


\(^{19}\) Byrne, Cosgrove, and Mehra, “Stereo based obstacle detection for an unmanned air vehicle”. 2006.
Stereo Vision

- On MAVs for a while now \(^{18,19}\)
  - generally too slow for fast flight

\(^{19}\) Byrne, Cosgrove, and Mehra, “Stereo based obstacle detection for an unmanned air vehicle”. 2006.
Stereo Vision

► On MAVs for a while now $^{18,19}$
  ► generally too slow for fast flight

Fast stereo vision:

$^{18}$Hrabar et al., “Combined optic-flow and stereo-based navigation of urban canyons for a UAV”. 2005.
$^{19}$Byrne, Cosgrove, and Mehra, “Stereo based obstacle detection for an unmanned air vehicle”. 2006.
Stereo Vision

- On MAVs for a while now \(^{18,19}\)
  - generally too slow for fast flight

Fast stereo vision:
- GPU stereo \(^{20}\)

\(^{19}\) Byrne, Cosgrove, and Mehra, “Stereo based obstacle detection for an unmanned air vehicle”. 2006.
Stereo Vision

- On MAVs for a while now\textsuperscript{18,19}
  - generally too slow for fast flight

Fast stereo vision:
- GPU stereo\textsuperscript{20}
- FPGA stereo\textsuperscript{21,22}

\textsuperscript{18}Hrabar et al., “Combined optic-flow and stereo-based navigation of urban canyons for a UAV”. 2005.
\textsuperscript{19}Byrne, Cosgrove, and Mehra, “Stereo based obstacle detection for an unmanned air vehicle”. 2006.
\textsuperscript{20}Yang and Pollefeys, “Multi-resolution real-time stereo on commodity graphics hardware”. 2003.
\textsuperscript{21}Honegger et al., “Real-time velocity estimation based on optical flow and disparity matching”. 2012.
\textsuperscript{22}Honegger, Oleynikova, and Pollefeys, “Real-time and Low Latency Embedded Computer Vision Hardware Based on a Combination of FPGA and Mobile CPU”. 2014.
Contributions

1. A novel, fast stereo algorithm for obstacle detection
2. High-speed control algorithms for integrating vision
3. A demonstration platform
Contributions

1. A novel, fast stereo algorithm for obstacle detection
Contributions

1. A novel, fast stereo algorithm for obstacle detection
2. High-speed control algorithms for integrating vision
Contributions

1. A novel, fast stereo algorithm for obstacle detection
2. High-speed control algorithms for integrating vision
3. A demonstration platform
Stereo vision
Block-Matching Stereo Vision

Left

Right
Block-Matching Stereo Vision

Left

Right
Block-Matching Stereo Vision

Left

Right
Block-Matching Stereo Vision

Left

Right

[Diagram showing stereo vision with a tree in the left and right frames, and a line connecting the two to illustrate depth perception.]
Block-Matching Stereo Vision

Left

Right
Block-Matching Stereo Vision

Left

Right
Block-Matching Stereo Vision

Left

Right
Block-Matching Stereo Vision

Left

Right

Grid for stereo vision with marked features in both the left and right images.
Block-Matching Stereo Vision

Left

Right
Block-Matching Stereo Vision

Left

Right
Block-Matching Stereo Vision

Left

Right

![Diagram showing block-matching stereo vision with a tree on the left and a smaller tree on the right, indicating depth perception.]
Block-Matching Stereo Vision

Left

Right
Block-Matching Stereo Vision

Left

Right
Block-Matching Stereo Vision

Left

Right
Issue: this search takes a long time.

- On flight hardware: **5-10 frames per second**
  - Quad core ARM, 1.7Ghz
  - 376x240 grayscale image

ODROID-U3 computer
(image courtesy Hardkernel co., Ltd.)
Issue: this search takes a long time.

- On flight hardware: 5-10 frames per second
  - Quad core ARM, 1.7Ghz
  - 376x240 grayscale image

10 fps: 1.2m / frame
Issue: this search takes a long time.

- On flight hardware: **5-10 frames per second**
  - Quad core ARM, 1.7Ghz
  - 376x240 grayscale image

10 fps: 1.2m / frame
120 fps: 0.1m / frame

ODROID-U3 computer
(image courtesy Hardkernel co., Ltd.)
Idea: Don’t do the search

Instead, ask: is this pixel block 10 meters away?
Pushbroom Stereo

Left

Right
Pushbroom Stereo

Left

Right
Pushbroom Stereo

Left

Right
Pushbroom Stereo

Left

Right
Aircraft is moving faster than almost anything in the environment
Pushbroom Stereo

- Aircraft is moving faster than almost anything in the environment
Pushbroom Stereo

- Aircraft is moving faster than almost anything in the environment

Detection area
Pushbroom Stereo

- Aircraft is moving faster than almost anything in the environment

- Detection area
Pushbroom Stereo

- Aircraft is moving faster than almost anything in the environment.
Visual Horizontal Invariance

**Issue:** Horizon exhibits substantial visual horizontal invariance.

- On the 5x5 pixel block level

![Diagram](image-url)
Filtering Visual Horizontal Invariance

What is different about these false-positives?

Strategy: Search for a second match at the disparity corresponding to distances >15 meters away.

In practice, calibration is not perfect, so search many possibilities near that region.
Filtering Visual Horizontal Invariance

What is different about these false-positives?

- They have another match nearby.

Strategy:

- Search for a second match at the disparity corresponding to distances > 15 meters away.

In practice, calibration is not perfect, so search many possibilities near that region.
Filtering Visual Horizontal Invariance

What is different about these false-positives?

- They have another match nearby.

**Strategy**: Search for a second match at the disparity corresponding to distances $> 15$ meters away.
Filtering Visual Horizontal Invariance

What is different about these false-positives?

- They have another match nearby.

**Strategy**: Search for a second match at the disparity corresponding to distances $> 15$ meters away.

- In practice, calibration is not perfect, so search many possibilities near that region
Filtering Visual Horizontal Invariance

What is different about these false-positives?

- They have another match nearby.

**Strategy**: Search for a second match at the disparity corresponding to distances > 15 meters away.

- In practice, calibration is not perfect, so search many possibilities near that region.
Pushbroom stereo implementation

120 frames per second

- Fully multithreaded
- Single-instruction multiple-data (ARM NEON SIMD)
- Leaves 1x computer available for control processing

ODROID-U3 computer
(image courtesy Hardkernel co., Ltd.)
Note: all flights have an onboard safety tether
False-Positive Benchmark

☐ = detection at 5 meters
False-Positive Benchmark

Benchmark against OpenCV’s block-matching stereo:
False-Positive Benchmark

Benchmark against OpenCV’s block-matching stereo:

- Walk on the ground, collecting 23,000+ frames
  - various outdoor environments and lighting conditions
False-Positive Benchmark

Benchmark against OpenCV’s block-matching stereo:

▶ Walk on the ground, collecting 23,000+ frames
  ▶ various outdoor environments and lighting conditions
▶ Run pushbroom stereo and OpenCV block-matching
False-Positive Benchmark

Benchmark against OpenCV’s block-matching stereo:

- Walk on the ground, collecting 23,000+ frames
  - various outdoor environments and lighting conditions
- Run pushbroom stereo and OpenCV block-matching
- Compute minimum 3D distance from pushbroom to BM stereo points
False-Positive Benchmark

Benchmark against OpenCV’s block-matching stereo:

- Walk on the ground, collecting 23,000+ frames
  - various outdoor environments and lighting conditions
- Run pushbroom stereo and OpenCV block-matching
- Compute minimum 3D distance from pushbroom to BM stereo points
False-Positive Benchmark

Benchmark against OpenCV’s block-matching stereo:

- Walk on the ground, collecting 23,000+ frames
  - various outdoor environments and lighting conditions
- Run pushbroom stereo and OpenCV block-matching
- Compute minimum 3D distance from pushbroom to BM stereo points
False-Positive Benchmark

Benchmark against OpenCV’s block-matching stereo:

- Walk on the ground, collecting 23,000+ frames
  - various outdoor environments and lighting conditions
- Run pushbroom stereo and OpenCV block-matching
- Compute minimum 3D distance from pushbroom to BM stereo points
False-Positive Benchmark

On over 23,000+ frames:

- Pushbroom stereo produces points within:

![Graph showing false-positive benchmark](image)
False-Positive Benchmark

On over 23,000+ frames:

- Pushbroom stereo produces points within:
  - 1.0 meters of StereoBM 71.2% of the time

![Graph showing fraction of pixels vs. separation in meters]

- Separation (meters)
  - 0 2 4 6
- Fraction of Pixels
  - 0 0.2 0.4 0.6 0.8 1
False-Positive Benchmark

On over 23,000+ frames:

- Pushbroom stereo produces points within:
  - 1.0 meters of StereoBM 71.2% of the time
  - 2.0 meters of StereoBM 81.0% of the time
False-Negative Benchmark

- "Opposite" of the false-positive approach: compute distance from BM stereo to pushbroom
- Run only on flight data (requires hand-labeling for StereoBM)
False-Negative Benchmark

- “Opposite” of the false-positive approach: compute distance from BM stereo to pushbroom
False-Negative Benchmark

- “Opposite” of the false-positive approach: compute distance from BM stereo to pushbroom
- Run only on flight data (requires hand-labeling for StereoBM)
False-Negative Benchmark

- “Opposite” of the false-positive approach: compute distance from BM stereo to pushbroom
- Run only on flight data (requires hand-labeling for StereoBM)
False-Negative Benchmark

- “Opposite” of the false-positive approach: compute distance from BM stereo to pushbroom
- Run only on flight data (requires hand-labeling for StereoBM)
False-Negative Benchmark

- “Opposite” of the false-positive approach: compute distance from BM stereo to pushbroom
- Run only on flight data (requires hand-labeling for StereoBM)
False-Negative Benchmark

- Pushbroom stereo misses points that Stereo BM detects by:

![Graph showing the fraction of pixels as a function of separation in meters. The graph has a y-axis labeled 'Fraction of Pixels' ranging from 0 to 1 and an x-axis labeled 'Separation (meters)' ranging from 0 to 6.]
False-Negative Benchmark

- Pushbroom stereo misses points that Stereo BM detects by:
  - 1.0 meters of StereoBM 67.6% of the time

![Graph showing separation vs fraction of pixels](image-url)
False-Negative Benchmark

- Pushbroom stereo misses points that Stereo BM detects by:
  - 1.0 meters of StereoBM 67.6% of the time
  - 2.0 meters of StereoBM 91.3% of the time
Onboard state estimation

Goal: GPS denied

- Start with an open source state estimator (Kalman filter)
- Add inputs for:
  - Barometric altimeter
  - Pitot tube airspeed sensor

Onboard state estimation

Goal: GPS denied
Onboard state estimation

Goal: GPS denied

- Start with an open source state estimator (Kalman filter)\textsuperscript{23}

Onboard state estimation

Goal: GPS denied

▶ Start with an open source state estimator (Kalman filter)\(^ {23}\)

▶ Add inputs for:

Onboard state estimation

Goal: GPS denied

- Start with an open source state estimator (Kalman filter)\textsuperscript{23}

- Add inputs for:
  - Barometric altimeter

Onboard state estimation

Goal: GPS denied

- Start with an open source state estimator (Kalman filter)\(^{23}\)

- Add inputs for:
  - Barometric altimeter
  - Pitot tube airspeed sensor

Onboard state estimation

Good estimation of:

- altitude
- roll
- pitch
- yaw
- forward speed
- climb rate
- angular rates

Limited ability to estimate:

- absolute $x$ and $y$ positions
- sufficient for pushbroom stereo
Onboard state estimation

Good estimation of:

- altitude

Limited ability to estimate:

- absolute \( x \) and \( y \) positions
- sufficient for pushbroom stereo
Onboard state estimation

Good estimation of:

- altitude
- roll
- pitch
- yaw
- forward speed
- climb rate

Limited ability to estimate:

- absolute x and y positions
- sufficient for pushbroom stereo
Onboard state estimation

Good estimation of:

- altitude
- roll
- pitch
Onboard state estimation

Good estimation of:

- altitude
- roll
- pitch
- yaw

Limited ability to estimate:

- absolute x and y positions
- sufficient for pushbroom stereo
Onboard state estimation

Good estimation of:

- altitude
- roll
- pitch
- yaw
- forward speed
Onboard state estimation

Good estimation of:

- altitude
- roll
- pitch
- yaw
- forward speed
- climb rate
Onboard state estimation

Good estimation of:

- altitude
- roll
- pitch
- yaw
- forward speed
- climb rate
- angular rates

Limited ability to estimate:

- absolute x and y positions
- sufficient for pushbroom stereo
Onboard state estimation

Good estimation of:
- altitude
- roll
- pitch
- yaw
- forward speed
- climb rate
- angular rates

Limited ability to estimate:
Onboard state estimation

Good estimation of:

- altitude
- roll
- pitch
- yaw
- forward speed
- climb rate
- angular rates

Limited ability to estimate:

- absolute x and y positions
Onboard state estimation

Good estimation of:

- altitude
- roll
- pitch
- yaw
- forward speed
- climb rate
- angular rates

Limited ability to estimate:

- absolute $x$ and $y$ positions
- sufficient for pushbroom stereo
26 MPH
26 MPH
147 ft
pitch/roll
Outline

Sensing:

- Pushbroom stereo for obstacle detection
- Inertial, airspeed, and barometric sensors for state estimation
Outline

Sensing:

- Pushbroom stereo for obstacle detection
Outline

Sensing:
- Pushbroom stereo for obstacle detection
- Inertial, airspeed, and barometric sensors for state estimation
Outline

Sensing:
  ▶ Pushbroom stereo for obstacle detection
  ▶ Inertial, airspeed, and barometric sensors for state estimation

Control:
Outline

Sensing:
- Pushbroom stereo for obstacle detection
- Inertial, airspeed, and barometric sensors for state estimation

Control:
- Trajectory libraries
Outline

Sensing:
- Pushbroom stereo for obstacle detection
- Inertial, airspeed, and barometric sensors for state estimation

Control:
- Trajectory libraries
- TVLQR feedback control
Trajectory Libraries

- Precomputed trajectories
- Choose trajectory to execute online
- Used on other robots for some time

Trajectory Libraries

- Precomputed trajectories
- Choose trajectory to execute online
- Used on other robots for some time


Dey et al., "Vision and Learning for Deliberative Monocular Cluttered Flight". 2015.

Trajectory Libraries

- Precomputed trajectories


Dey et al., "Vision and Learning for Deliberative Monocular Cluttered Flight". 2015.

Trajectory Libraries

- Precomputed trajectories
- Choose trajectory to execute online


Dey et al., "Vision and Learning for Deliberative Monocular Cluttered Flight". 2015.

Trajectory Libraries

▶ Precomputed trajectories
▶ Choose trajectory to execute online
▶ Used on other robots for some time $^{24,25,26}$

$^{26}$Majumdar and Tedrake, “Funnel Libraries for Robust Realtime Feedback Motion Planning”. 2016.
Building trajectories
Building trajectories
A model-based approach

Model-based design allows:

▶ Optimization of trim conditions, trajectories, and controllers
▶ Easy conversion to other airframes
▶ Safety verification
A model-based approach

Model-based design allows:

▶ Optimization of trim conditions, trajectories, and controllers
A model-based approach

Model-based design allows:

- Optimization of trim conditions, trajectories, and controllers
- Easy conversion to other airframes
A model-based approach

Model-based design allows:

- Optimization of trim conditions, trajectories, and controllers
- Easy conversion to other airframes
- Safety verification
Aircraft model

Nonlinear model: $\dot{x} = f(x, u)$
Aircraft model

Nonlinear model: \( \dot{x} = f(x, u) \)

- state vector
Aircraft model

Nonlinear model: \( \dot{x} = f(x, u) \)

- state vector
- 12 states \( (x) \)
  - \( x, y, \) and \( z \)
  - roll, pitch, and yaw
  - derivatives of those 6 states
Aircraft model

Nonlinear model: $\dot{x} = f(x, u)$

- state vector
- 12 states ($x$)
  - $x$, $y$, and $z$
  - roll, pitch, and yaw
  - derivatives of those 6 states
- control vector
Aircraft model

Nonlinear model: $\dot{x} = f(\ x, \ u)$

- state vector
- 12 states ($x$)
  - $x, y, \text{ and } z$
  - roll, pitch, and yaw
  - derivatives of those 6 states

- control vector
- 3 inputs ($u$)
  1. left control surface
  2. right control surface
  3. throttle
Aircraft model

\[ \dot{x} = f \left( \begin{array}{c} x \\ u \end{array} \right) \]

state

control input
Aircraft model

\[ \dot{x} = f(x, u) \]

state
control input

flat-plate dynamics
Aircraft model

\[ \dot{x} = f \left( x, u \right) \]

state
control input

flat-plate dynamics
Aircraft model

\[ \dot{x} = f \left( x, u \right) \]

flat-plate dynamics
Control about a trim condition

Straight and level flight:
Control about a trim condition

Straight and level flight:

\[ \dot{x} = f( x, u ) \]
\[ x = [x \ y \ z \ \phi \ \theta \ \psi \ \dot{x} \ \dot{y} \ \dot{z} \ \dot{\phi} \ \dot{\theta} \ \dot{\psi}]^T \]

- roll
- pitch
- yaw
Control about a trim condition

Straight and level flight:

\[
\dot{x} = f(x, u)
\]

\[
x = \begin{bmatrix}
x
y
z
\phi
\theta
\psi
\dot{x}
\dot{y}
\dot{z}
\dot{\phi}
\dot{\theta}
\dot{\psi}
\end{bmatrix}^T
\]

roll → pitch ↔ yaw

\[
\ddot{x} = \begin{bmatrix}
\ddot{x}
\ddot{y}
\ddot{z}
\ddot{\phi}
\ddot{\theta}
\ddot{\psi}
\end{bmatrix}^T
\]

accelerations
Searching for a trim condition

state and control input

\[ \text{find } x_0, u_0 \]
Searching for a trim condition

state and control input

\[
\text{find } x_0, u_0
\]

\[
\text{s.t.}
\]

accelerations = 0, \quad \Leftarrow 6 \text{ nonlinear constraints}
Searching for a trim condition

state and control input

\[ \text{find } \mathbf{x}_0, \mathbf{u}_0 \]

s.t.

\[ \begin{align*}
\text{accelerations} &= 0, \quad \iff 6 \text{ nonlinear constraints} \\
\mathbf{u}_0 &\geq \mathbf{u}_{\text{min}}, \quad \iff 3 \text{ linear constraints}
\end{align*} \]
Searching for a trim condition

state and control input
find $x_0, u_0$

s.t.

accelerations = 0, $\iff$ 6 nonlinear constraints
$u_0 \geq u_{min}$, $\iff$ 3 linear constraints
$u_0 \leq u_{max}$, $\iff$ 3 linear constraints
Searching for a trim condition

state and control input

\[
\text{find } x_0, u_0
\]

\[s.t.\]

accelerations = 0, \iff 6 nonlinear constraints

\[
u_0 \geq u_{\text{min}}, \iff 3 \text{ linear constraints}
\]

\[
u_0 \leq u_{\text{max}}, \iff 3 \text{ linear constraints}
\]

giving \( x_0 \) and \( u_0 \)
Stabilizing the trim condition

Using standard nonlinear control techniques:

\[ \bar{x} = x - x_0 \]

\[ \bar{u} = -K \bar{x} \]

\[ u = \bar{u} + u_0 \]
Stabilizing the trim condition

Using standard nonlinear control techniques:

\[
\bar{x} = x - x_0
\]

\[
\bar{u} = -K \bar{x}
\]

\[
u = \bar{u} + u_0
\]

With our model, we can linearize about the trim condition

► (Taylor approximate our nonlinear model)
Stabilizing the trim condition

Using standard nonlinear control techniques:

\[
\begin{align*}
\bar{x} &= x - x_0 \\
\tilde{u} &= -K \bar{x} \\
u &= \tilde{u} + u_0
\end{align*}
\]

With our model, we can linearize about the trim condition

- (Taylor approximate our nonlinear model)

giving: \( \dot{x} = A\bar{x} + B\tilde{u} \)
Stabilizing the trim condition

Using standard nonlinear control techniques:

\[
\bar{x} = \underbrace{x}_{\text{current state}} - \underbrace{x_0}_{\text{desired state}}
\]

\[
\bar{u} = -K \bar{x}
\]

\[
\underbrace{u}_{\text{control input}} = \bar{u} + u_0
\]

With our model, we can linearize about the trim condition

- (Taylor approximate our nonlinear model)

giving: \[ \dot{\bar{x}} = A\bar{x} + B\bar{u} \]

allowing us to use linear control
Manual / auto
Autonomous Takeoff

Set $\dot{z} > 0$:

(don’t change the gains)
Autonomous Takeoff

Set $\dot{z} > 0$:

(don’t change the gains)

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & \dot{\phi} & \dot{\theta} & \dot{\psi} & \ddot{x} & \ddot{y} & \ddot{z} & \ddot{\phi} & \ddot{\theta} & \ddot{\psi} \end{bmatrix}^T$$

forward velocity
climbing

accelerations
Autonomous Takeoff

Set $\dot{z} > 0$:

\[(\text{don’t change the gains})\]

\[\dot{x} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & \dot{\phi} & \dot{\theta} & \dot{\psi} & \ddot{x} & \ddot{y} & \ddot{z} & \dddot{\phi} & \dddot{\theta} & \dddot{\psi} \end{bmatrix}^T\]

forward velocity
climbing
giving $x_0$ and $u_0$
Dynamic Maneuvers
Dynamic Maneuvers

Not a trim condition
Dynamic Maneuvers

Two options for finding an open-loop trajectory:
Dynamic Maneuvers

Two options for finding an open-loop trajectory:

1. Trajectories from manual flights
Dynamic Maneuvers

Two options for finding an open-loop trajectory:
1. Trajectories from manual flights
2. Trajectory optimization
Trajectories from manual flights
Trajectory optimization
Trajectory optimization

- Optimize over $x(t)$ and $u(t)$ to find an open loop trajectory
Knife-edge: $x$, $y$, and $z$ tracking

Dotted vertical lines: trajectory change
Knife-edge: roll, pitch, and yaw

![Graphs showing roll, pitch, and yaw over time.](image)
Knife-edge: control actions

![Diagram showing actual and planned control surface deflections for left and right surfaces over time](image-url)
Outline

Sensing:
- Pushbroom stereo for obstacle detection
- Inertial, airspeed, and barometric sensors for state estimation

Control:
- Trajectory libraries
- TVLQR feedback control
Outline

Sensing:
- Pushbroom stereo for obstacle detection
- Inertial, airspeed, and barometric sensors for state estimation

Control:
- Trajectory libraries
- TVLQR feedback control
- Online planning
Picking a good trajectory online

1. Is current trajectory in collision?
2. If yes, for each trajectory:
   2.1 Compute minimum distance between time-sampled trajectory and point cloud
   2.2 Reject if penetrates the ground
3. Execute trajectory with maximum distance to point cloud

▶ Makes a decision within 18.9 ms
Picking a good trajectory online

1. Is current trajectory in collision?
Picking a good trajectory online

1. Is current trajectory in collision?
2. If yes, for each trajectory:
   2.1 Compute minimum distance between time-sampled trajectory and point cloud
   2.2 Reject if penetrates the ground
3. Execute trajectory with maximum distance to point cloud

▶ Makes a decision within 18.9ms
Picking a good trajectory online

1. Is current trajectory in collision?
2. If yes, for each trajectory:
   2.1 Compute minimum distance between time-sampled trajectory and point cloud
Picking a good trajectory online

1. Is current trajectory in collision?

2. If yes, for each trajectory:
   2.1 Compute minimum distance between time-sampled trajectory and point cloud
   2.2 Reject if penetrates the ground
Picking a good trajectory online

1. Is current trajectory in collision?
2. If yes, for each trajectory:
   2.1 Compute minimum distance between time-sampled trajectory and point cloud
   2.2 Reject if penetrates the ground
3. Execute trajectory with maximum distance to point cloud
Picking a good trajectory online

1. Is current trajectory in collision?
2. If yes, for each trajectory:
   2.1 Compute minimum distance between time-sampled trajectory and point cloud
   2.2 Reject if penetrates the ground
3. Execute trajectory with maximum distance to point cloud

▶ Makes a decision within 18.9ms
Experiments
Experimental plan

(autonomous modes in blue)

Takeoff from catapult launcher → Control (no throttle) → Clear cable → Climb → Cruise / avoid

Manual landing
Autonomous takeoff from launcher
Autonomous obstacle avoidance
Analysis

Used a simple trajectory library:

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>Type</th>
<th>Length</th>
<th>Produced</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Straight</td>
<td>Trim</td>
<td></td>
<td>Model</td>
</tr>
<tr>
<td>2</td>
<td>Climb</td>
<td>Trim</td>
<td></td>
<td>Model</td>
</tr>
<tr>
<td>3</td>
<td>Takeoff (no throttle)</td>
<td>Trim</td>
<td></td>
<td>Model</td>
</tr>
<tr>
<td>4</td>
<td>Gentle left</td>
<td>Trim</td>
<td></td>
<td>Model</td>
</tr>
<tr>
<td>5</td>
<td>Gentle right</td>
<td>Trim</td>
<td></td>
<td>Model</td>
</tr>
<tr>
<td>6</td>
<td>Left jog</td>
<td>Dynamic</td>
<td>2.45s</td>
<td>Flight data</td>
</tr>
<tr>
<td>7</td>
<td>Right jog</td>
<td>Dynamic</td>
<td>2.49s</td>
<td>Flight data</td>
</tr>
</tbody>
</table>
$x$, $y$, and $z$ tracking
Roll, pitch, and yaw
Add a chase plane:

* Autonomous plane
* Manual chase plane

[Image of a field with two planes labeled as Autonomous plane and Manual chase plane]
Aggregate Analysis

Over 16 successful flights:

- 1.5 km flown autonomously
- 7,951 stereo matches detected
- 163 trajectories executed
- 131 seconds in autonomous mode
- with an average speed of 12.1 m/s (27 mph)
Aggregate Analysis

Over 16 successful flights:

- 1.5 km flown autonomously
- 7,951 stereo matches detected
- 163 trajectories executed
- 131 seconds in autonomous mode
- with an average speed of 12.1 m/s (27 mph)
3 environments:
Obstacles (farther)

Obstacles (closer)
### Failure Analysis

<table>
<thead>
<tr>
<th>Obstacle type</th>
<th>Total flights</th>
<th>Successes</th>
<th>Success ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artificial</td>
<td>4</td>
<td>4</td>
<td>100%</td>
</tr>
<tr>
<td>Pair of trees</td>
<td>4</td>
<td>4</td>
<td>100%</td>
</tr>
<tr>
<td>Many trees</td>
<td>18</td>
<td>8</td>
<td>44%</td>
</tr>
</tbody>
</table>
Failure Analysis

<table>
<thead>
<tr>
<th>Obstacle type</th>
<th>Total flights</th>
<th>Successes</th>
<th>Success ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artificial</td>
<td>4</td>
<td>4</td>
<td>100%</td>
</tr>
<tr>
<td>Pair of trees</td>
<td>4</td>
<td>4</td>
<td>100%</td>
</tr>
<tr>
<td>Many trees</td>
<td>18</td>
<td>8</td>
<td>44%</td>
</tr>
</tbody>
</table>

- Failures were split between vision and control equally:
## Failure Analysis: Vision

<table>
<thead>
<tr>
<th>Failure Type</th>
<th>Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vision failures</strong></td>
<td>5</td>
</tr>
<tr>
<td>Failed to see obstacle</td>
<td>1</td>
</tr>
<tr>
<td>Poor calibration</td>
<td>2</td>
</tr>
<tr>
<td>No video data / unknown vision failure</td>
<td>2</td>
</tr>
</tbody>
</table>
Failure Analysis: Vision

Failed to see obstacle a combination of:

1. Low contrast obstacles (grey leaves over sky)
2. High angular rate occludes obstacle until it is closer than 10m
## Failure Analysis: Control

<table>
<thead>
<tr>
<th>Failure Type</th>
<th>Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control failures</td>
<td>5</td>
</tr>
<tr>
<td>Insufficiently rich maneuver library</td>
<td>2</td>
</tr>
<tr>
<td>Trajectory initial state</td>
<td>2</td>
</tr>
<tr>
<td>Loss of control</td>
<td>1</td>
</tr>
</tbody>
</table>
Insufficiently rich maneuver library

- No "turn 90°" trajectory available
Trajectory initial state

- Known issue: our trajectories only start with level flight.

- Potentially surprising: failure when aircraft is already rolled in the direction of future travel.

An example:
1. Start rolled left
2. Choose to execute a left turn
3. First control action is: hard right roll
Trajectory initial state

► **Known issue**: our trajectories only start with level flight
Trajectory initial state

- **Known issue**: our trajectories only start with level flight
- **Potentially surprising**: failure when aircraft is *already* rolled in the direction of future travel.
Trajectory initial state

- **Known issue**: our trajectories only start with level flight
- **Potentially surprising**: failure when aircraft is *already* rolled in the direction of future travel.

An example:
Trajectory initial state

- **Known issue**: our trajectories only start with level flight
- **Potentially surprising**: failure when aircraft is *already* rolled in the direction of future travel.

An example:

1. Start rolled **left**
Trajectory initial state

- **Known issue**: our trajectories only start with level flight
- **Potentially surprising**: failure when aircraft is already rolled in the direction of future travel.

An example:
1. Start rolled **left**
2. Choose to execute a **left** turn
Trajectory initial state

- **Known issue**: our trajectories only start with level flight
- **Potentially surprising**: failure when aircraft is *already* rolled in the direction of future travel.

An example:

1. Start rolled **left**
2. Choose to execute a **left** turn
3. First control action is:
Trajectory initial state

- **Known issue**: our trajectories only start with level flight
- **Potentially surprising**: failure when aircraft is already rolled in the direction of future travel.

An example:

1. Start rolled **left**
2. Choose to execute a **left** turn
3. First control action is: **hard right roll**
Incorrect action at trajectory start

Recovery from impact

Time (s)
Actual
Planned
Impact with obstacle
Time (s)
125.5 126 126.5 127 127.5
Left control surface deflection (deg)
-40
-30
-20
-10
0
10
20
30
40
Actual
Planned
Incorrect action at trajectory start
Impact event
Moving forward

Trajectory libraries:
- Multiple starting states in trajectory library
- Verification for switching trajectories like $^{27}$

$^{27}$Majumdar and Tedrake, “Funnel Libraries for Robust Realtime Feedback Motion Planning”. 2016.
Moving forward

Wind:
Moving forward

Wind:
  - Onboard wind sensing \(^{28}\)

\(^{28}\)Xue et al., “Refraction wiggles for measuring fluid depth and velocity from video”. 2014.
Moving forward

Wind:
- Onboard wind sensing ²⁸
- Control through wind ²⁹, ³⁰

²⁸Xue et al., “Refraction wiggles for measuring fluid depth and velocity from video”. 2014.
²⁹Majumdar and Tedrake, “Robust Online Motion Planning with Regions of Finite Time Invariance”. 2012.
Moving forward

Pushbroom stereo:
Moving forward

Pushbroom stereo:
  ▶ Search multiple depths
Moving forward

Pushbroom stereo:

- Search multiple depths
  - Check for false positives

Small OpenCL capable GPUs have just entered the market
Moving forward

Pushbroom stereo:

- Search multiple depths
  - Check for false positives
  - Track obstacles
Moving forward

Pushbroom stereo:

- Search multiple depths
  - Check for false positives
  - Track obstacles
  - Check along a planned trajectory
Moving forward

Pushbroom stereo:

- Search multiple depths
  - Check for false positives
  - Track obstacles
  - Check along a planned trajectory
- GPU implementation
Moving forward

Pushbroom stereo:
- Search multiple depths
  - Check for false positives
  - Track obstacles
  - Check along a planned trajectory
- GPU implementation
  - Small OpenCL capable GPUs have just entered the market
Moving forward

Safe operation of small autonomous aircraft in clutter with:

- Fast, agile flight
- Provably safe control with perception in the loop
- Deep integration of accurate vision systems
Moving forward

Safe operation of small autonomous aircraft in clutter with:

- Fast, agile flight
Moving forward

Safe operation of small autonomous aircraft in clutter with:

- Fast, agile flight
- \textit{Provably} safe control with perception in the loop
Moving forward

Safe operation of small autonomous aircraft in clutter with:

- Fast, agile flight
- *Provably* safe control with perception in the loop
- Deep integration of accurate vision systems
Moving forward

**Flight experiments are expensive**
Moving forward

**Flight experiments are expensive**

Can we build models that include vision and control?
Moving forward

**Flight experiments are expensive**

Can we build models that include vision and control?
  
  ▶ systematically find and correct failure modes for:
Moving forward

**Flight experiments are expensive**

Can we build models that include vision and control?
- systematically find and correct failure modes for:
  - vision
Moving forward

**Flight experiments are expensive**

Can we build models that include vision and control?

- systematically find and correct failure modes for:
  - vision
  - control
Moving forward

Flight experiments are expensive

Can we build models that include vision and control?

- systematically find and correct failure modes for:
  - vision
  - control
  - closed loop system
Moving forward

Flight experiments are expensive

Can we build models that include vision and control?
  ▶ systematically find and correct failure modes for:
    ▶ vision
    ▶ control
    ▶ closed loop system

Good answers for control,
Moving forward

Flight experiments are expensive

Can we build models that include vision and control?

- systematically find and correct failure modes for:
  - vision
  - control
  - closed loop system

Good answers for control, more to do for vision systems
Contributions

1. Pushbroom stereo for high-speed obstacle detection
Contributions

1. Pushbroom stereo for high-speed obstacle detection
2. Control algorithms for integrating (1) in the loop
Contributions

1. Pushbroom stereo for high-speed obstacle detection
2. Control algorithms for integrating (1) in the loop
3. Demonstration of the fastest MAV flying in complex obstacles with only onboard sensing and computation to date
Everything is open source:
Everything is open source:

- Flight code:
  - github.com/andybarry
Everything is open source:

- Flight code:
  - [github.com/andybarry](https://github.com/andybarry)

- Our lab’s simulation and analysis environment (Drake)
  - [drake.mit.edu](https://drake.mit.edu)
Acknowledgements

A huge number of people helped make this possible.

**Advisor**: Russ Tedrake  
**Thesis committee**: Bill Freeman, Nick Roy

**Labmates**
- Ani Majumdar
- Pete Florence
- John Carter
- Tim Jenks
- Benoit Landry
- Andy Marchese
- Hongkai Dai
- Joseph Moore
- Zack Jackowski
- the entire Robot Locomotion Group

**Collaborators**
- Helen Oleynikova
- Jacob Izraelevitz
- John Rom
- Nadya Peek
- Gabriel Klabin
- Dave Barrett
- Mark Chang
- MURI team

**CSAIL**
- Ron Wiken
- Mieke Moran
- Bryt Bradley
- Mark Pearrow
- Adam Conner-Simons
- Abby Abazorius
- Kathy Bates

Fourth East, EC Houseteam, Olin scope team

Mom, Dad, Katya, and Jenny