Robotics at Compile Time:

Optimizing Robotics Algorithms With C++'s Compile-Time Features

STEPHEN BRAWNER

Cppcon
The C++ Conference

2023

October 01 - 06

About me

- Robotics Software Consultant based in Portland, OR
- Chief Roboticist, FLX Solutions
- PhD in CS, specializing in Robotics and AI
- Past Clients: Open Robotics, Picknik, Verb Surgical, Magnopus, Formant, Amazon
- Industries: Robotics Software, Solar Technologies, Surgical Robotics, General Robotics, TV & Film,

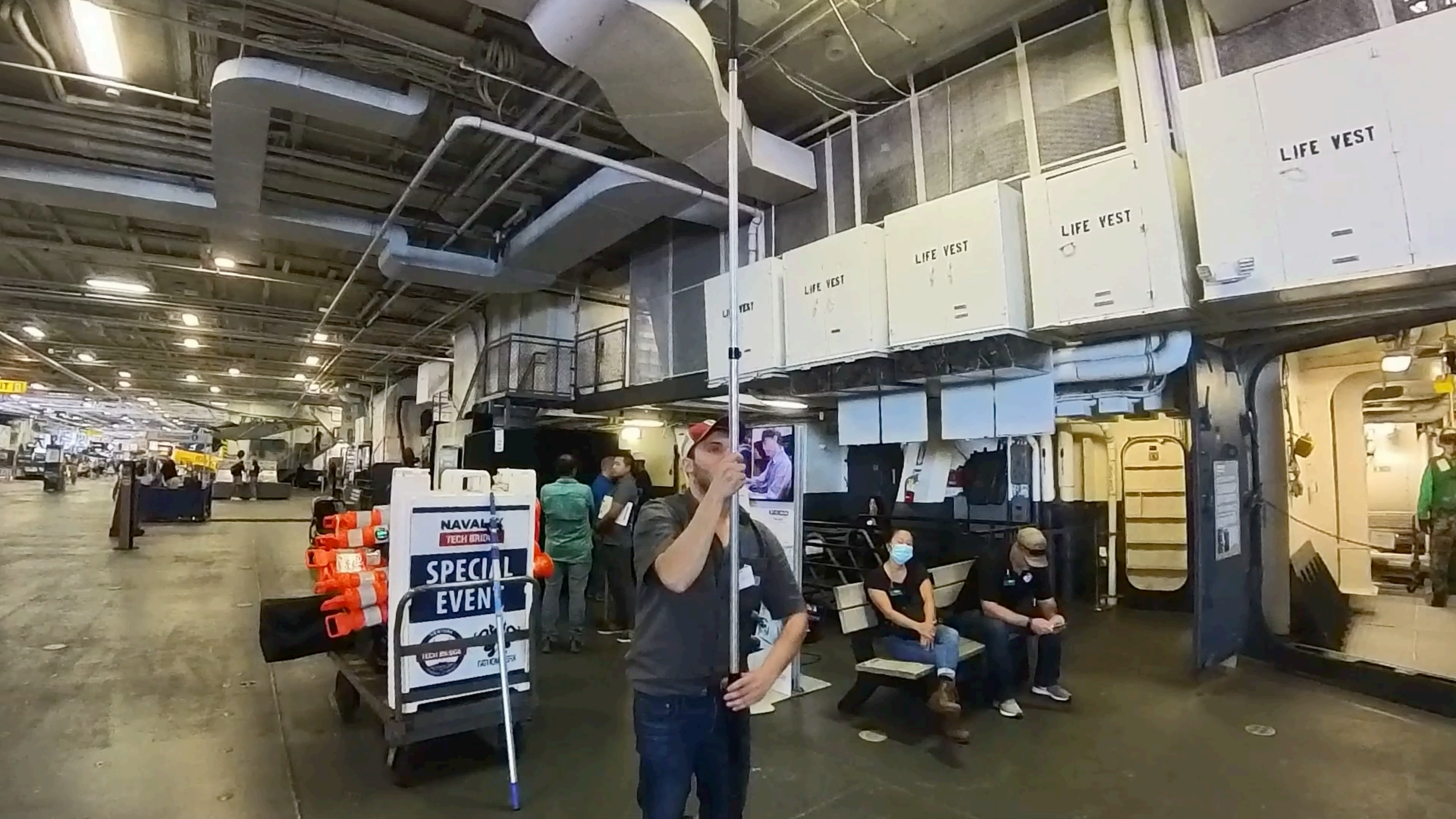
Software in Robotics

Robot Agnostic

- Designed to be used across most types of robots
- Parses robot description from a plain-text file format
- Requires lots of run-time decisions
- Generally maintained by a corporate entity
  - ROS and ROS 2 ecosystem (Open Robotics/ Intrinsic)
  - Drake (TRI)
  - Ignition and Gazebo (Open Robotics/Intrinsic)
  - Isaac (Nvidia)

Robot Specific

- Most software at robotics hardware companies
- Maintained by themselves
- Can be written with targeted robots and platforms in mind
- Generally required for real-time controls


LIFE VEST
LIFE VEST
LIFE VEST
LIFE VEST
LIFE VEST
SPECIAL
EVENT

Robotics Hardware Development

- Requirements known well ahead of time
- Mechanical and electrical designs are locked-in years in advance
- Production robots probably won’t grow limbs or develop new senses
- Software and firmware can target specific board architectures

Boston Dynamics Spot robot, announced June 2016, first released late 2019

Robotics Firmware/Software Development

Robotics developers ship software much more frequently than they ship robot hardware

Robots Developed	~1/year
Software builds	> 1/year

Robotics Firmware/Software Development

Goal:

- Write generalizable software
- Portable across robots and platforms
- As if written specifically for one robot (0 cost abstractions)

Robotics Firmware/Software Development

```
#include "robots/my_robot_specs.h"

#include "robot_application.h"

int main(){
    application<robots::my_robot>::run();
}
```

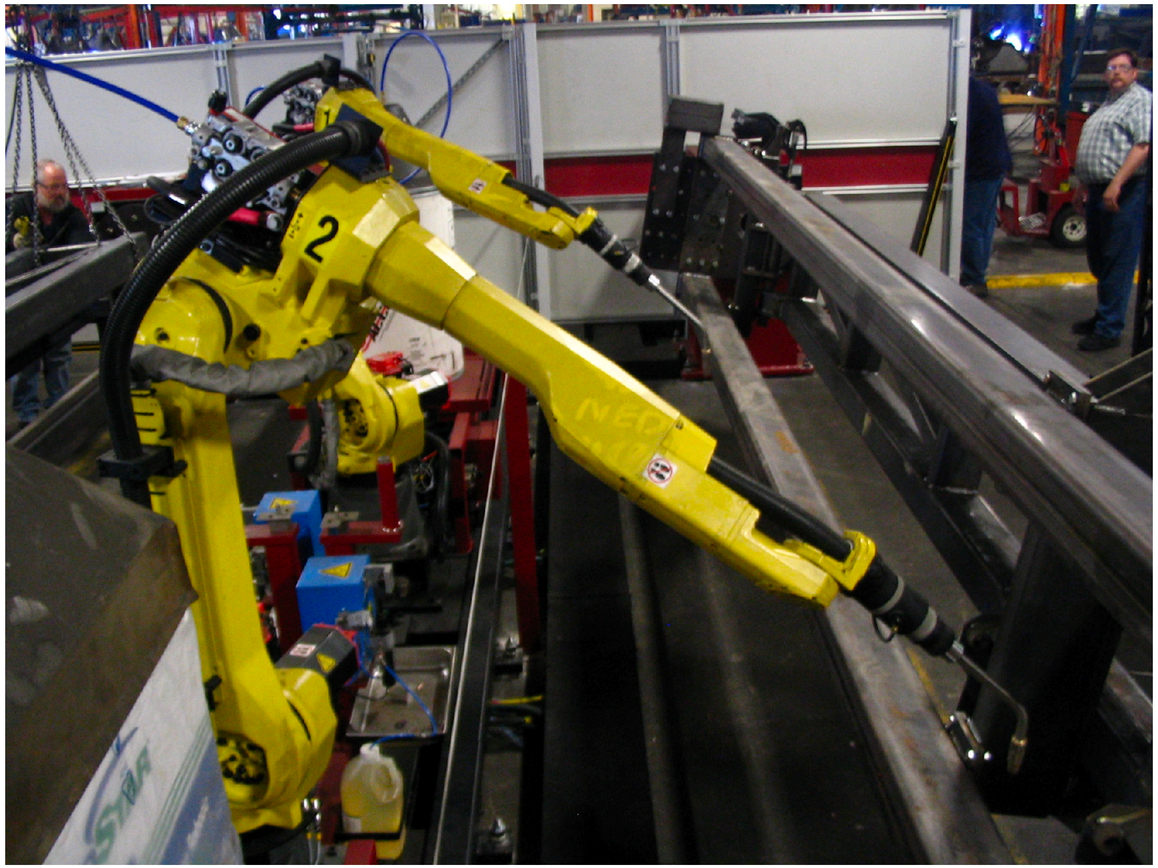
Real-Time Software

Deterministic software that produces the same result for a known input

Delivers the correct result at the correct time

Real-time software

Classification	Missed Deadlines	Example
Hard	Program fault	Surgical Robotics
Firm	Result is treated as worthless	Robotic Assembly Line
Soft	Performance Degrades	Audio/Video processing



Real-Time Software In Robotics

- Requires real-time OS or run bare metal
- General concepts to avoid in real-time software
  - Dynamic Memory Allocation
  - Exceptions
  - Device I/O
  - Locking primitives that block
- Optimize for CPU and memory budgets, missed deadlines

https://design.ros2.org/articles/realtime_background.html

Why C++ is Great for Real-Time Robotics

- isocpp and `--Wpedantic`
- Requirements traceability
  - One can write code that avoids allocation, exceptions, I/O and locks, etc
- Strong type safety let’s you use tooling to validate your organization’s software and its possible states
- Performance

What could be better

- Safety critical software may not be able to use the STL
- Standardized guarantees around allocations, locks, and I/O in the STL
- Perhaps a `constmem` or `noallocate` code decorators

Compile-time Optimizations in Robotics Software

Compile Time Optimizations in Robotics Software

```
#include "my_robot_specs.h"

#include "robot_application.h"

int main(){
    application<my_robot>::run();
}
```

Stack-Allocated Ring Buffer Example

```
template <typename T, size_t MaxSize>
class ring_buffer {
    aligned_memory<T, Size> data_;
    size_t head_{0U};
    size_t tail_{0U};
   …
};
```

How Hard Could it Be? - David Stone
https://www.youtube.com/watch?v=I8QJLGI0GOE

Stack-Allocated Ring Buffer Example

```
template <typename T, size_t MaxSize>
class ring_buffer {
    using size_type = smallest_size_type_for<MaxSize>::type;
    aligned_memory<T, MaxSize> data_;
    size_type head_{0U};
    size_type tail_{0U};
   …
};
```

Stack-Allocated Ring Buffer Example

```
template <size_t Size>
using smallest_size_type_for = std::conditional<
    (Size < std::numeric_limits<uint8_t>::max()),
    uint8_t,
    typename smallest_size_type_for_16<Size>::type>::type type;
```

Stack-Allocated Ring Buffer Example

```
template <typename T, size_t MaxSize>
class ring_buffer {
…
   constexpr void push_back(const T& value){
      if (size() < MaxSize){
          data_.store(head_, value);
          head_ = (head_ + 1U) % MaxSize;
      } else {
          // Drop or overwrite?
      }
 }
};
```

Stack-Allocated Ring Buffer Example

```
template <typename T, size_t MaxSize, typename FillBehavior>
class ring_buffer {
  FillBehavior fill_behavior_;
…
public:
  constexpr ring_buffer(FillBehavior fill_behavior = FillBehavior{});

  constexpr void push_back(const T& value){
      fill_behavior_.push_back(head_, tail_, data_, value);
};
```

Stack-Allocated Ring Buffer Example

```
enum class FillBehaviorType {Overwrite, Ignore};

template <typename T, size_t MaxSize, FillBehaviorType F>
class RingBuffer {
   using FillBehaviorT = FillBehavior<F>::type;
};

template <FillBehaviorType F>
struct FillBehavior;

template <>
struct FillBehavior<FillBehaviorType::Overwrite>{…};

template <>
struct FillBehavior<FillBehaviorType::Ignore>{…};
```

Robotics Control Problem

Robotics Control Problem

- FLX Solutions FLX BOT: a snake-like robot for inspection
- Need to move the end effector in cartesian space according to joystick input
  - Read encoders and write to motor driver > 100 Hz
  - Code is executed on embedded chip with 100s KB of memory
  - No self-collisions, control oscillations or drop outs

Forward Kinematics

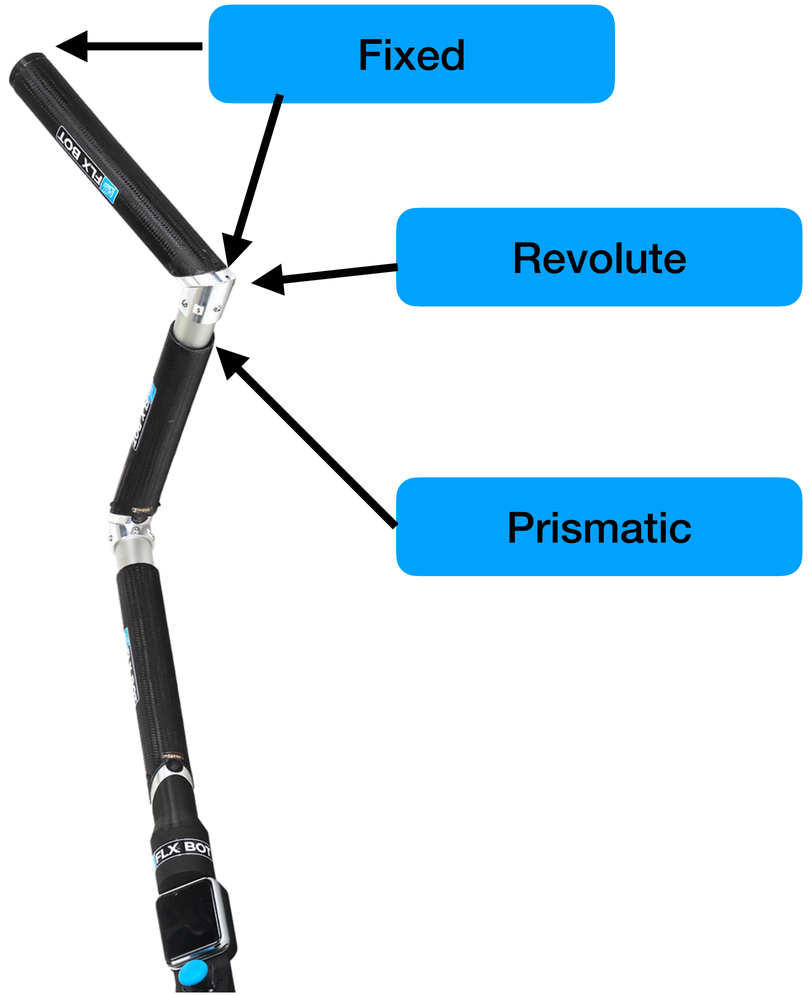
- The problem of computing the position and orientation of links on the robot as a function of the robot's joint positions

Link	Each rigid component of the robot
Joint	Mechanisms between links that generate the motion

Forward Kinematics

- Types of Joints

Revolute	Rotation about a fixed axis
Linear/Prismatic	Translates along fixed axis
Fixed	Describes additional reference frames



Forward Kinematics

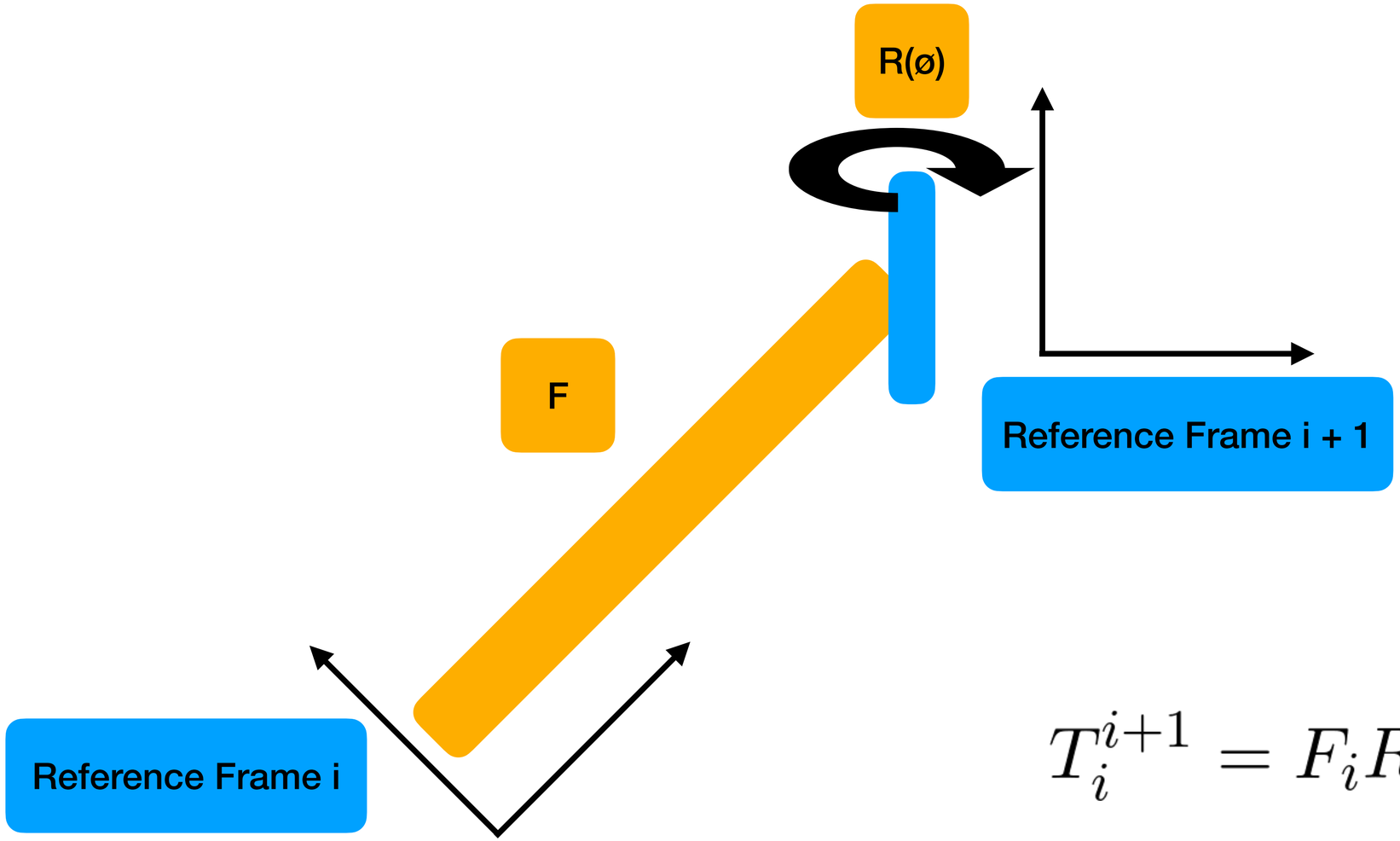
- Described as a geometric transformation

P	Point
T	Transform
R	Rotation
U	Translation

$$\mathbf{P_2} = T_1^2 \mathbf{P_1}$$

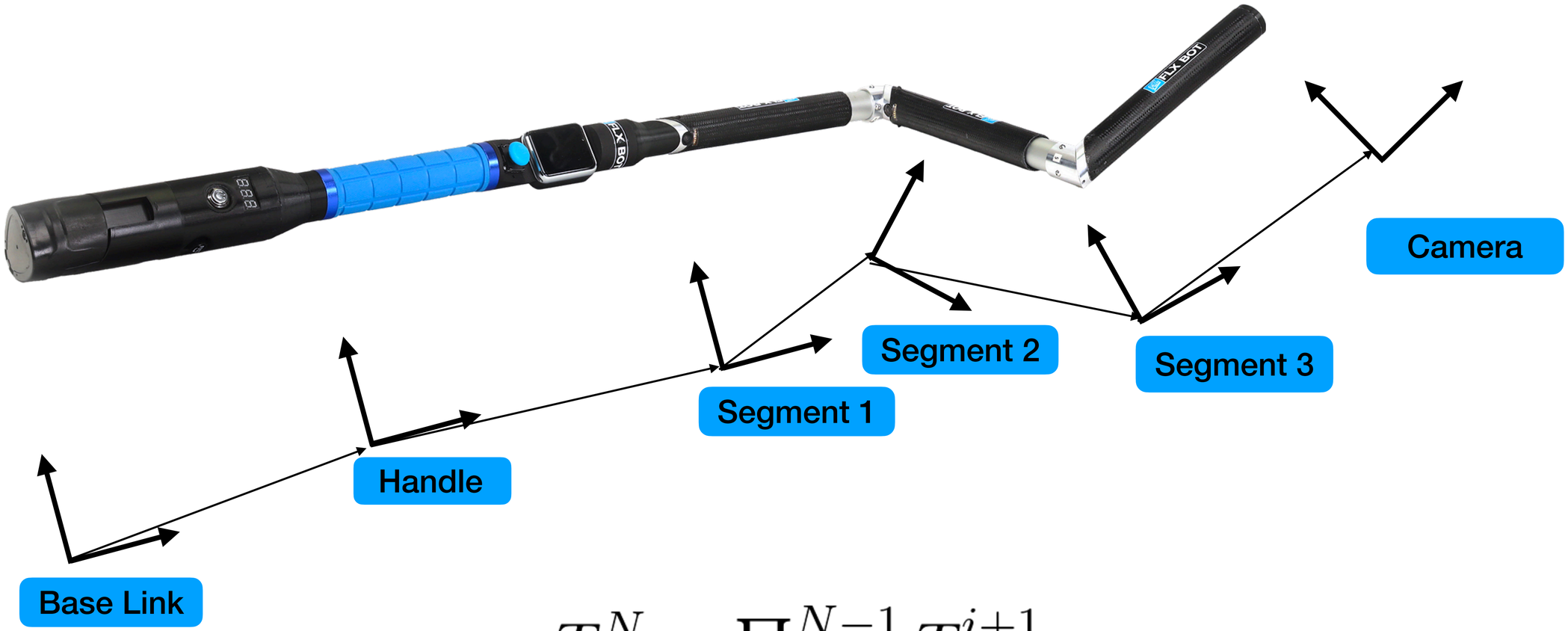
$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & u_x \\ r_{21} & r_{22} & r_{23} & u_y \\ r_{31} & r_{32} & r_{33} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$$

Forward Kinematics



$$T_i^{i+1} = F_i R(\theta_i)$$

Forward Kinematics



$$T_0^N = \prod_0^{N-1} T_i^{i+1}$$

One way of solving this problem…

```
using Transform = Eigen::Isometry3d;
using Joint = std::variant<Fixed, Revolute, Prismatic>;

class Fixed {
  Transform fixed_;

public:
  Fixed(const Transform& fixed) : fixed_{fixed} {}

  Transform pose(double /* position */) const {return fixed_;}
};
```

One way of solving this problem…

```
class Revolute {
  Transform fixed_;
  Eigen::Vector3d axis_;
…
  Transform pose(double position) const {
    return fixed_ * Eigen::AngleAxisd(position, axis_);
  }
};
```

One way of solving this problem…

```
class Prismatic {
…
  Transform pose(double position) const {
    return fixed_ * Eigen::Translation3d(position * axis_);
  }
};
```

Requisite godbolt.org demo

- https://godbolt.org/z/5jM9sa3vY

Reducing Duplication With Metaprogramming

```
enum class JointType {Fixed, Prismatic, Revolute};

template <JointType JT>
class Joint {
…
    Transform pose(double position) const {
        return fixed_ * calculate_joint_pose<JT>(position, axis_);
    }
};

template <JointType JT>
auto calculate_joint_pose(double position, const Eigen::Vector3d& axis){
    if constexpr(JT == JointType::Fixed) {return Eigen::Isometry3d::Identity(); }
    else if constexpr (JT == JointType::Prismatic) {return Eigen::Translation3d(position * axis); }
    else if constexpr (JT == JointType::Revolute) {return Eigen::AngleAxisd(position, axis); }
}
```

Reducing Duplication With Metaprogramming

```
template <JointType JT, typename FixedTransform>
class Joint {
    FixedTransform fixed_;
…
    Joint(const FixedTransform& fixed) : fixed_{fixed} {}

    Transform pose(double position) const {
        return fixed_ * calculate_joint_pose<JT>(position, axis_);
}
};
```

- With explicit classes and runtime dispatch
  - https://godbolt.org/z/9YMP3aqs5
- With better compile-time dispatch
  - https://godbolt.org/z/fKd9h4ofd

Handling Joint Limits

```
template <JointType JT, typename FixedTransform>
class Joint {
   Limits limits_;
…
    Transform pose(double position) const {
        const auto clamped_position = std::clamp(position, limits_.lower, limits_.upper);
        return fixed_ * calculate_joint_pose<JT>(position, axis_);
    }
};
```

Handling Joint Limits

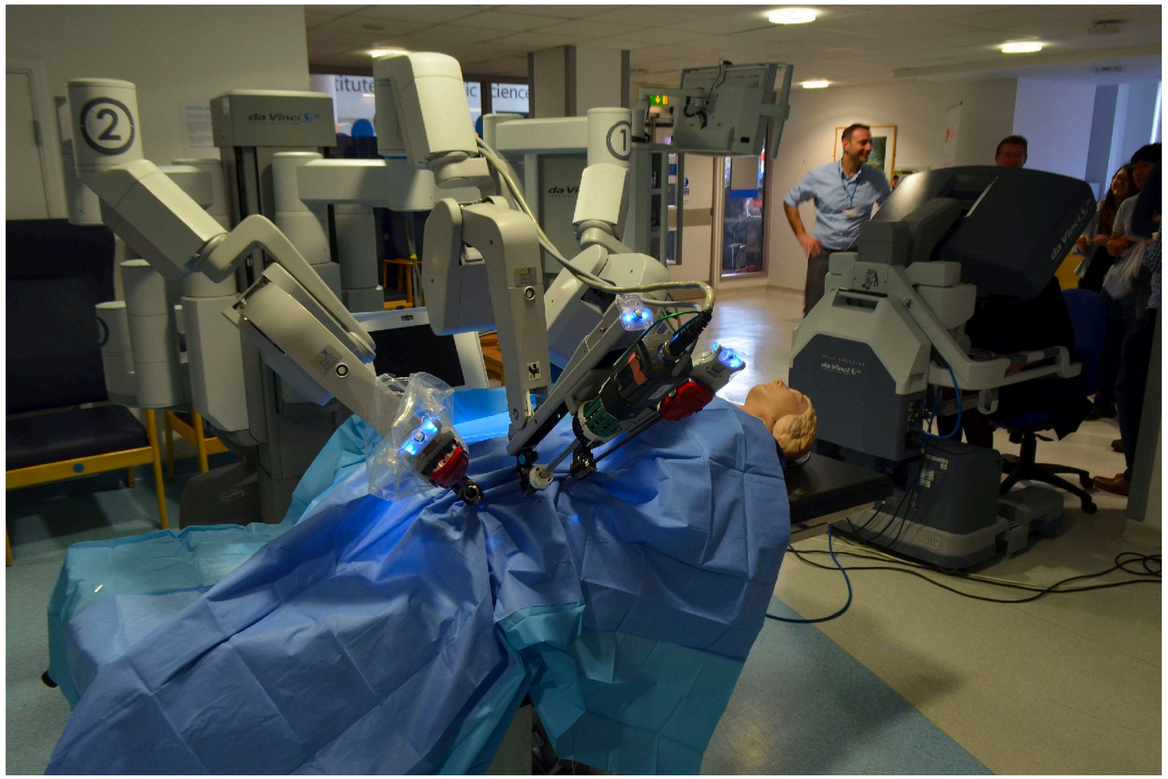
```
template <JointType JT, typename FixedTransform, typename LimitsT>
class Joint {
    LimitsT limits_;
…
    Joint(FixedTransform fixed, LimitsT limits)
      : fixed_{fixed}, limits_{limits} {}

    Transform pose(double position) const {
        const auto clamped_position = limits_(position);
        return fixed_ * calculate_joint_pose<JT>(position, axis_);
    }
};
```

Handling Joint Limits

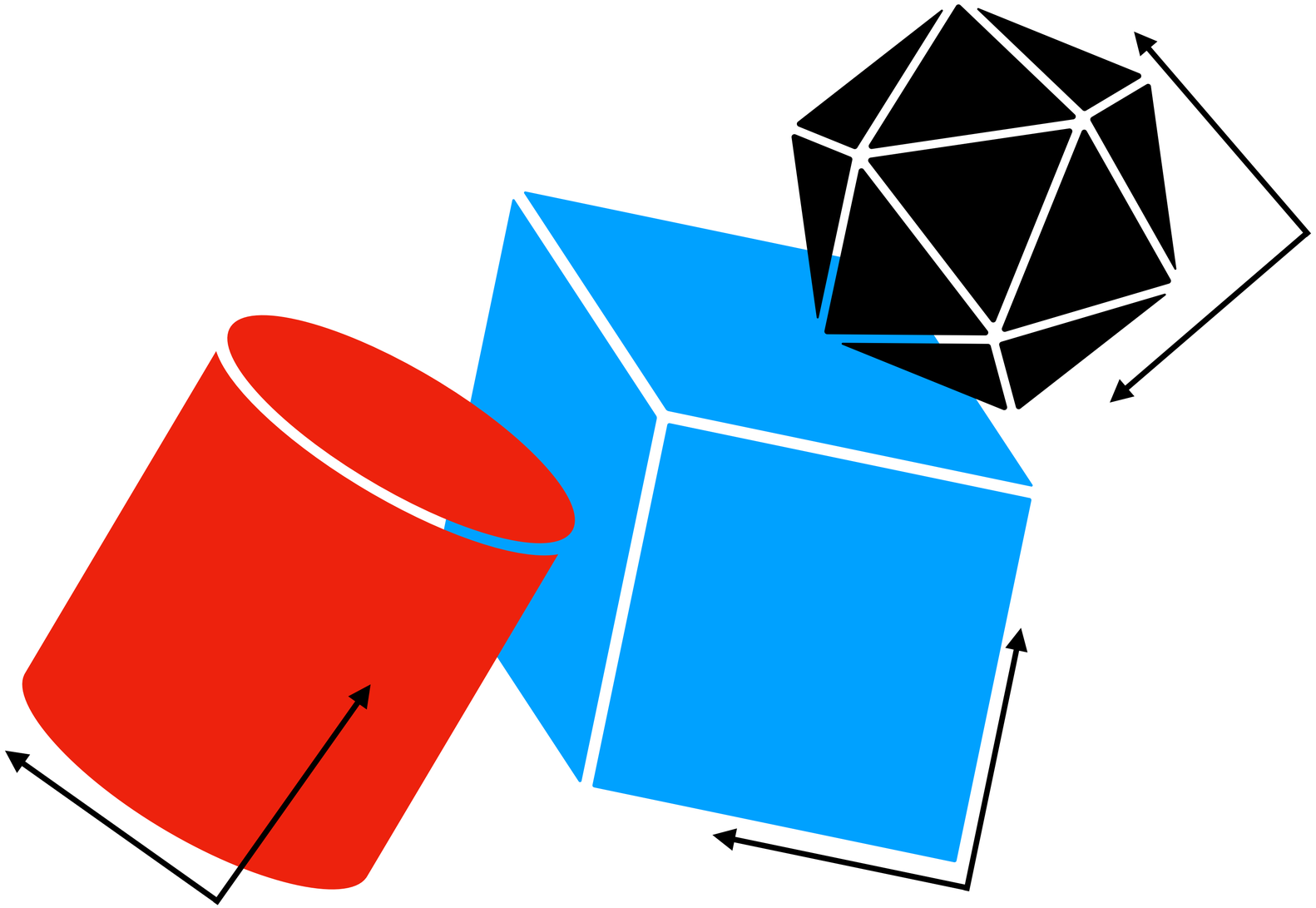
```
constexpr auto InfiniteLimits = [](double value){return value;};

class WrappedLimits{
…
    double operator()(double value){
        return modulo_with_offset(value, limits_.upper - limits_.lower, limits_.lower);
    }
};

class ClampedLimits{
…
    template <typename T>
    double operator()(double value){
        return std::clamp(value, limits_.lower, limits_.upper);
    }
};
```

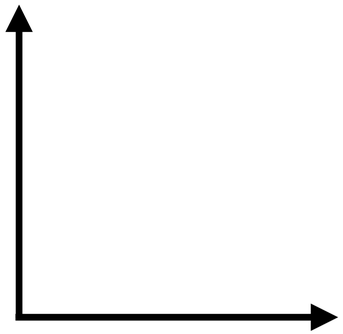
Collision Detection

- Need to decide in real-time: is the robot colliding with itself
- Each physical link is represented by one or more collision shapes
- O(n^2) or O(n log n)



Collisions with Shapes





Collision Detection at Runtime With Ranges

```
auto is_in_collision =
  [](const ShapeWithPose& shape1, const ShapeWithPose& shape2){
    ...
};

auto is_in_collision_tuple =
  [](const std::tuple<ShapeWithPose, ShapeWithPose>& shapes){
    return is_in_collision(std::get<0>(shapes), std::get<1>(shapes));
};

template <typename Range>
bool is_robot_in_collision(const Range& shapes){
    return std::ranges::cartesian_product(shapes, shapes)
      | std::ranges::transform(is_in_collision_tuple)
      | std::ranges::none_of;
}
```

Collision Detection at Compile Time

```
bool is_in_collision(
  const BoxWithPose& shape1, const SphereWithPose& shape2){...}
…

template <typename...Args>
bool is_robot_in_collision(const std::tuple<Args...>& shapes_with_poses){
    return std::apply([](const auto...args){
        return is_robot_in_collision(args...);}, shapes);
}
```

Collision Detection at Compile Time

```
template <typename...Shapes>
bool is_robot_in_collision(const Shapes&... shapes){
    return (is_shape_in_collision(shapes, shapes) ||...);
}

template <typename Shape1, typename...OtherShapes>
bool is_shape_in_collision(
        const Shape1& shape1, OtherShapes...other_shapes){
    return (is_in_collision(shape1, other_shapes) ||...);
}
```

Collision Detection

```
template
bool is_

    retu
}

template
bool is_
    retu
}
```

Reducing the collision checks to N(N-2)/2 is left to the viewer at home

Summary

- Hardware companies (not just robotics) often build and ship code much more often than they ship new hardware
- Utilizing c++ meta programming techniques, the compiler can see and optimize much more of your code than your poor overworked development team
- Leverage c++’s many compile-time features to build rich, descriptive types
- Thank you