spanny

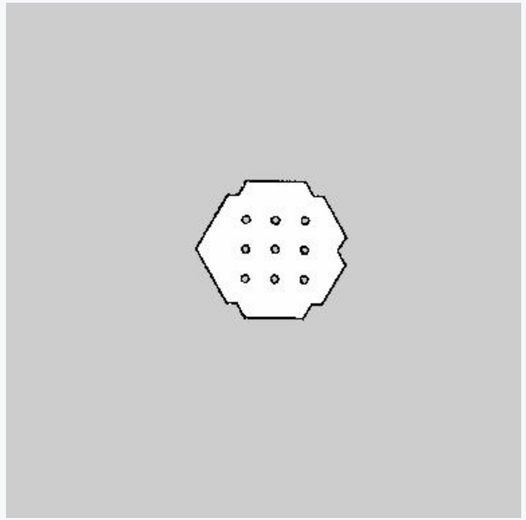
abusing mdspan is within arms reach

October 5, 2023

Griswald Brooks
Senior Robotics Engineer
griswald.brooks@picknik.ai

mdspan

```
// Allocate the map
std::array map_storage{...};
// Create a 2D view of the map
auto occupancy_map = std::mdspan(map_storage.data(), 384, 384);
// Update the map
for (std::size_t i = 0; i != occupancy_map.extent(0); i++) {
    for (std::size_t j = 0; j != occupancy_map.extent(1); j++) {
        occupancy_map[i, j] = get_occupancy_from_laser_scan(...)
    }
}
```



github.com/griswaldbrooks/spanny

Policy Injection

```
template <
  class T,
  class Extents,
  class LayoutPolicy = std::layout_right,
  class Accessor = std::default_accessor
>
class mdspan;
```

github.com/griswaldbrooks/spanny

Bin Accessor Policy

```
template <int nbins>
struct bin_checker {
  using element_type = tl::expected<bin_state, std::string>;
  using reference = tl::expected<bin_state, std::string> const&;
  using data_handle_type = robot_arm*;

  inline static element_type const no_bin = tl::make_unexpected("Invalid bin");
  mutable element_type recent_ = tl::make_unexpected("Bin has not been accessed");

  reference access(data_handle_type arm, std::ptrdiff_t offset) const {
    // Return an error if the offset is out of bounds
    if (offset < 0 || offset >= nbins) return no_bin;
    // Set the bin state in a member of the accessor policy
    recent_ = arm->is_bin_occupied(static_cast<int>(offset));
    // This is because we return a reference
    return recent_;
  }
};
```

Bin Accessor Policy

```
struct bin_checker {
  using element_type = bin_state;
  using reference = std::future<bin_state>;
  using data_handle_type = robot_arm*;

  reference access(data_handle_type ptr, std::ptrdiff_t offset) const {
    return std::async([=]{
      return bin_state{ptr->is_bin_occupied(static_cast<int>(offset)),
                       offset_to_coord(static_cast<int>(offset))};
    });
  }
};
```

Bin Accessor Policy

```
struct bounds_checked_layout_policy {
  template <class Extents>
  struct mapping : stdex::layout_right::mapping<Extents> {
    using base_t = stdex::layout_right::mapping<Extents>;
    using base_t::base_t;
    std::ptrdiff_t operator()(auto... idxs) const {
      [&]<size_t... Is>(std::index_sequence<Is...>) {
        if (((idxs < 0 || idxs > this->extents().extent(Is)) || ...)) {
          throw std::out_of_range("Invalid bin index");
        }
      }(std::make_index_sequence<sizeof...(idxs)>{});
      return this->base_t::operator()(idxs...);
    }
  };
};
```

Async Beer View

```
int main() {
  auto arm = robot_arm{"/dev/ttyACM0", 9600};
  auto bins = bin_view(&arm);
  while(true) {
    std::vector<std::future<bin_state>> futures;
    for (auto i = 0u; i < bins.extent(0); ++i)
      for (auto j = 0u; j < bins.extent(1); ++j)
          futures.push_back(bins(i, j));

    for (auto const& future : futures) future.wait();

    for (auto& future : futures) print_state(future.get());

    std::cout << "====================" << std::endl;
  }
  return 0;
}
```

github.com/griswaldbrooks/spanny

Demo Time

github.com/griswaldbrooks/spanny

Thank you!

Special Thanks to
Daisy Hollman and Tyler Weaver
github.com/griswaldbrooks/spanny

Slides generated using Elsie