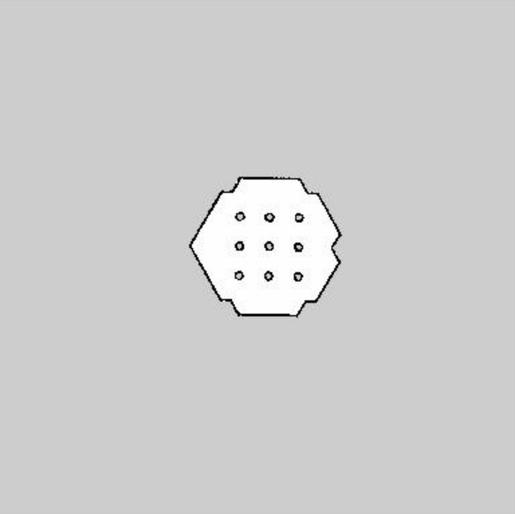
spanny: mdspan

```
// 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(...)
    }
}
```



spanny: policy injection

```
using bin_view =
std::mdspan<class T, class Extents, class LayoutPolicy, class Accessor>;
```

spanny: bin accessor

```
using bin_view =
std::mdspan<class T, class Extents, class LayoutPolicy, bin_checker>;

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))};
    });
  }
};
```

spanny: bin layout

```
using bin_view =
std::mdspan<class T, class Extents, bin_layout, bin_checker>;

struct bin_layout {
  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...);
    }
  };
};
```

spanny: beer view

```
using bin_view =
std::mdspan<bin_state, six_pack, bin_layout, bin_checker>;

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;
}
```

Demo Time

Thanks

Griswald Brooks | github.com/griswaldbrooks/spanny