Image Inference (Experiment 2)

Preprocessing

```
# Read in the participant data.
data_0 = read_csv(file.path(human_path, "data.csv"), quote="~")

# Read in the MTurk results file.
mturk_results = read_csv(file.path(human_path, "mturk_results.csv"),
                         col_names=TRUE) %>%
  mutate(Answer.surveycode=substr(Answer.surveycode, 6,
                                  length(Answer.surveycode))) %>%
  filter(AssignmentStatus=="Approved")

# Check for invalid survey codes.
invalid = sum(!(mturk_results$Answer.surveycode %in% data_0$unique_id))
if (invalid != 0) { stop("There's an invalid survey code.") }

# Extract the trial information for each participant and save it.
age = c()
for (p in 1:length(mturk_results$Answer.surveycode)) {
  # Find the database entry that contains the experiment code that this
  # participant submitted.
  data_1 = data_0 %>%
    filter(unique_id==mturk_results$Answer.surveycode[p]) %>%
    filter(nchar(results)==min(nchar(results)))

  # Extract and store this participant's age.
  age = c(age, as.integer(fromJSON(data_1$results)$subject_information$age))

  # Extract the experiment code that this participant submitted.
  experiment_code = substr(fromJSON(data_1$results)$catch_trials$experiment_code, 6, 13)

  # Find the database entry that contains the trial information for this
  # participant.
  data_2 = data_0 %>%
    filter(unique_id==experiment_code) %>%
    filter(nchar(results)==max(nchar(results)))

  # Convert the JSON string to a data frame and refactor.
  data_3 = fromJSON(data_2$results)$trials %>%
    select(-trial_num) %>%
    mutate(participant=as.character(p-1)) %>%
    select(participant, map, coords)

  # Save the trial information for this participant.
```

```
  write_csv(data_3, file.path(human_path, paste(p-1, ".csv", sep="")))
}
```

Model Comparison

Here we pass in participant-generated paths (N=40; M=38.25 years, SD=11.02 years) to our model and a baseline model and compute the posterior probability of each model generating each participant-generated path. We then divide the posterior probability from our model by that of the baseline model to compute the Bayes factor. A Bayes factor greater than one indicates our model explains participant judgments better; a Bayes factor less than one indicates the baseline model explains participant judgments better.

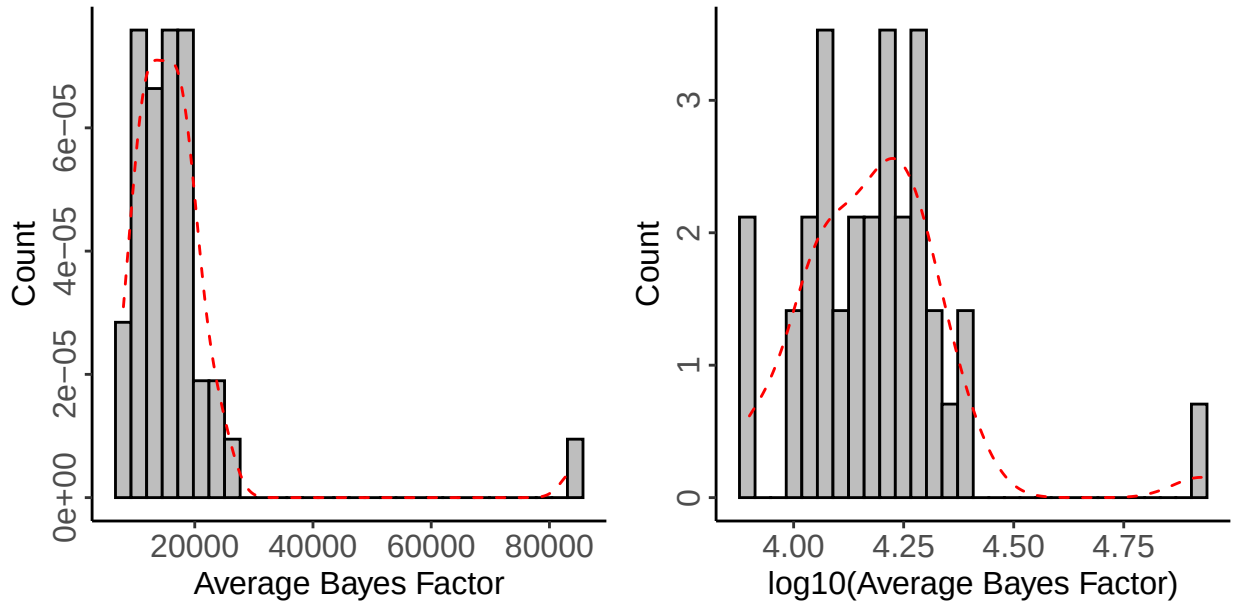
With a Bayes factor for each path for every participant, we then average them so that each participant has a single mean Bayes factor. First, we run a t-test comparing these mean Bayes factors against one to validate (from an NHST perspective) whether or not our model outperforms the baseline model.

```
##
##  One Sample t-test
##
## data:  data_5$mean_bayes_factor
## t = 9.1024, df = 39, p-value = 1.712e-11
## alternative hypothesis: true mean is greater than 1
## 95 percent confidence interval:
##  13800.74      Inf
## sample estimates:
## mean of x
##  16935.33
```

Next, we take a look at the distribution of mean Bayes factors among our 40 participants, both in standard scale and in log scale.

```
plot_0 = data_5 %>%
  ggplot(aes(x=mean_bayes_factor)) +
  geom_histogram(aes(y=stat(density)), fill="grey", color="black") +
  geom_density(color="red", linetype="dashed") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        axis.text=element_text(size=12),
        axis.text.y=element_text(angle=90, hjust=0.5),
        axis.title=element_text(size=12)) +
  xlab("Average Bayes Factor") +
  ylab("Count")

plot_1 = data_5 %>%
  mutate(log_mean_bayes_factor=log10(mean_bayes_factor)) %>%
  ggplot(aes(x=log_mean_bayes_factor)) +
  geom_histogram(aes(y=stat(density)), fill="grey", color="black") +
  geom_density(color="red", linetype="dashed") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        axis.text=element_text(size=12),
        axis.text.y=element_text(angle=90, hjust=0.5),
        axis.title=element_text(size=12)) +
```

```
  xlab("log10(Average Bayes Factor)") +
  ylab("Count")

plot_2 = plot_grid(plot_0, plot_1, nrow=1)
plot_2
```



Comparing participant-generated paths with participant inferences from Experiment 1

Here we test whether the goals and entrances from participant-generated paths have predictive power over inferences participants made in Experiment 1.

Preprocessing

First, we set up the paths to the data from Experiment 1.

```
# Set up the path to the participant data.
experiment_1_human_path = "data/experiment_1/human/data_0"
```

```
# Set up the path to the model data.
experiment_1_model_path = "data/experiment_1/model/predictions/Manhattan"
```

Next, we preprocess the goal inference data from Experiment 1.

```
# Read in the preprocessed data.
data_6 = read_csv(file.path(experiment_1_human_path, "data.csv"))

# Select and normalize the goal judgments.
data_7 = data_6 %>%
  select(unique_id, map, A, B, C) %>%
  gather(goal, human, A, B, C) %>%
  left_join(do(., summarize(group_by(., unique_id, map),
                            total_human=sum(human)))) %>%
  mutate(human=human/total_human) %>%
  select(-total_human)

# Define the bootstrap function for the bootstrap statistic.
compute_mean = function(data, indices) {
  return(mean(data[indices]))
}

# Define the bootstrap function to simulate the data.
compute_bootstrap = function(data) {
  # Run the simulations.
  simulations = boot(data=data,
                     statistic=compute_mean,
                     R=10000)

  return(boot.ci(simulations, type="bca")$bca)
}

# Compute the bootstrapped 95% CIs.
set.seed(seed)
ci = data.frame()
for (m in unique(data_7$map)) {
  # Filter the current map.
  data_8 = data_7 %>%
    filter(map==m)

  # Compute the bootstrap for each dependent measure.
  bootstrap_A = compute_bootstrap(filter(data_8, goal=="A")$human)
  bootstrap_B = compute_bootstrap(filter(data_8, goal=="B")$human)
  bootstrap_C = compute_bootstrap(filter(data_8, goal=="C")$human)

  # Store the bootstrapped 95% CIs for this pair.
  ci = rbind(ci, data.frame(map=rep(m, 3),
                            goal=c("A", "B", "C"),
                            lower=c(bootstrap_A[4],
                                    bootstrap_B[4],
                                    bootstrap_C[4]),
                            upper=c(bootstrap_A[5],
                                    bootstrap_B[5],
```

```
                                    bootstrap_C[5])))
}

# Read in the goal predictions.
model_0 = read_csv(file.path(experiment_1_model_path, "goal_predictions.csv"))

# Perform some basic preprocessing.
model_1 = model_0 %>%
  rename(model=probability)

# Merge the participant judgments with the bootstrapped 95% CIs and the model
# predictions.
data_9 = data_7 %>%
  group_by(unique_id) %>%
  ungroup() %>%
  group_by(map, goal) %>%
  summarize(mean_human=mean(human)) %>%
  ungroup() %>%
  left_join(ci) %>%
  left_join(model_1)
```

Then, we preprocess the entrance inference data from Experiment 1.

```
# Select and normalize the entrance judgments.
data_10 = data_6 %>%
  filter(!grepl("ND", map)) %>%
  select(unique_id, map, `1`, `2`, `3`) %>%
  gather(entrance, human, `1`, `2`, `3`) %>%
  filter(human!=-1) %>%
  left_join(do(., summarize(group_by(., unique_id, map),
                            total_human=sum(human)))) %>%
  mutate(human=human/total_human) %>%
  select(-total_human)

# Define the bootstrap function for the bootstrap statistic.
compute_mean = function(data, indices) {
  return(mean(data[indices]))
}

# Define the bootstrap function to simulate the data.
compute_bootstrap = function(data) {
  # Run the simulations.
  simulations = boot(data=data,
                     statistic=compute_mean,
                     R=10000)

  return(boot.ci(simulations, type="bca")$bca)
}

# Compute the bootstrapped 95% CIs.
set.seed(seed)
ci = data.frame()
for (m in unique(data_10$map)) {
```

```
  # Filter the current map.
  data_11 = data_10 %>%
    filter(map==m)

  # Compute the bootstrap for each dependent measure.
  bootstrap_1 = compute_bootstrap(filter(data_11, entrance=="1")$human)
  bootstrap_2 = compute_bootstrap(filter(data_11, entrance=="2")$human)
  if (length(unique(data_11$entrance)) == 3) {
    bootstrap_3 = compute_bootstrap(filter(data_11, entrance=="3")$human)
  }

  # Store the bootstrapped 95% CIs for this pair.
  if (length(unique(data_11$entrance)) == 3) {
    ci = rbind(ci, data.frame(map=rep(m, 3),
                              entrance=c("1", "2", "3"),
                              lower=c(bootstrap_1[4],
                                      bootstrap_2[4],
                                      bootstrap_3[4]),
                              upper=c(bootstrap_1[5],
                                      bootstrap_2[5],
                                      bootstrap_3[5])))
  }
  else {
    ci = rbind(ci, data.frame(map=rep(m, 2),
                              entrance=c("1", "2"),
                              lower=c(bootstrap_1[4],
                                      bootstrap_2[4]),
                              upper=c(bootstrap_1[5],
                                      bootstrap_2[5])))
  }
}

# Read in the entrance predictions.
model_2 = read_csv(file.path(experiment_1_model_path,
                             "entrance_predictions.csv"))

# Stitch the entrance mapping to the predictions and do some preprocessing.
model_3 = model_2 %>%
  right_join(read_csv(file.path(experiment_1_model_path,
                                "entrance_mapping.csv"))) %>%
  mutate(probability=ifelse(is.na(probability), 0, probability)) %>%
  select(-entrance) %>%
  rename(entrance=number, model=probability) %>%
  mutate(entrance=as.character(entrance))

# Merge the participant judgments with the bootstrapped 95% CIs and the model
# predictions.
data_12 = data_10 %>%
  group_by(unique_id) %>%
  ungroup() %>%
  group_by(map, entrance) %>%
  summarize(mean_human=mean(human)) %>%
  ungroup() %>%
```

```
  left_join(ci) %>%
  left_join(model_3)
```

Lastly, we preprocess the participant-generated path data from Experiment 2.

```
# Import the participant data.
data_14 = tibble()
for (i in 1:nrow(data_5)) {
  # Stitch together the path to this participant's data.
  data_path = paste(human_path, "/", i-1, ".csv", sep="")

  # Import this participant's data.
  data_14 = data_14 %>%
    rbind(read_csv(data_path, col_types="dcc"))
}

# Rearrange the data such that the maps are ordered alphabetically within each
# participant.
data_15 = data_14 %>%
  arrange(participant, map)

# Construct a mapping with the door locations.
doors = list(
  # DX_DX_0
  list(c(5, 1), c(1, 5)),
  # DX_NX_0
  list(c(5, 1), c(5, 9)),
  # DX_PX_0
  list(c(4, 1), c(9, 7)),
  # DX_UN_0
  list(c(5, 9), c(1, 3)),
  # ND_DX_0
  list(c(9, 5)),
  # ND_DX_1
  list(c(9, 3)),
  # ND_NX_0
  list(c(9, 5)),
  # ND_NX_1
  list(c(9, 3)),
  # ND_PX_0
  list(c(9, 3)),
  # ND_PX_1
  list(c(9, 4)),
  # ND_UN_0
  list(c(9, 7)),
  # NX_DX_0
  list(c(9, 5), c(7, 9), c(1, 5)),
  # NX_NX_0
  list(c(3, 1), c(9, 4), c(6, 9)),
  # NX_PX_0
  list(c(9, 6), c(6, 9), c(1, 3)),
  # NX_UN_0
  list(c(9, 5), c(7, 9), c(1, 5)),
```

```
  # PX_DX_0
  list(c(5, 1), c(7, 9), c(1, 4)),
  # PX_NX_0
  list(c(9, 5), c(7, 9), c(1, 6)),
  # PX_PX_0
  list(c(6, 1), c(6, 9), c(1, 3)),
  # PX_UN_0
  list(c(9, 4), c(9, 8), c(1, 5)),
  # UN_DX_0
  list(c(5, 1), c(1, 5)),
  # UN_NX_0
  list(c(7, 1), c(1, 7)),
  # UN_PX_0
  list(c(9, 6), c(6, 9)),
  # UN_UN_0
  list(c(9, 5), c(5, 9))
)

# Restructure the participant data.
data_18 = tibble()
for (p in 1:length(unique(data_15$participant))) {
  for (m in 1:length(unique(data_15$map))) {
    # Select a participant and map.
    data_16 = data_15 %>%
      filter(participant==unique(data_15$participant)[p],
             map==unique(data_15$map)[m])

    # Extract the x- and y-coordinates from their corresponding indices.
    indices_x = seq(1, nchar(data_16$coords), 4)
    indices_y = seq(3, nchar(data_16$coords), 4)
    x = c()
    y = c()
    for (i in 1:length(indices_x)) {
      x = c(x, as.integer(substr(data_16$coords, indices_x[i], indices_x[i])))
      y = c(y, as.integer(substr(data_16$coords, indices_y[i], indices_y[i])))
    }

    # Find out which door corresponds to the start of this path.
    diff = c()
    for (door in doors[[m]]) {
      diff = c(diff, abs(x[1]-door[1])+abs(y[1]-door[2]))
    }
    x = c(doors[[m]][[which(diff==min(diff))]][1], x)
    y = c(doors[[m]][[which(diff==min(diff))]][2], y)

    # Set up the new structure of the participant data.
    data_17 = data.frame(
      participant=unique(data_15$participant)[p],
      map=unique(data_15$map)[m],
      x=x,
      y=y,
      time=(1:length(x))-1
    )
```

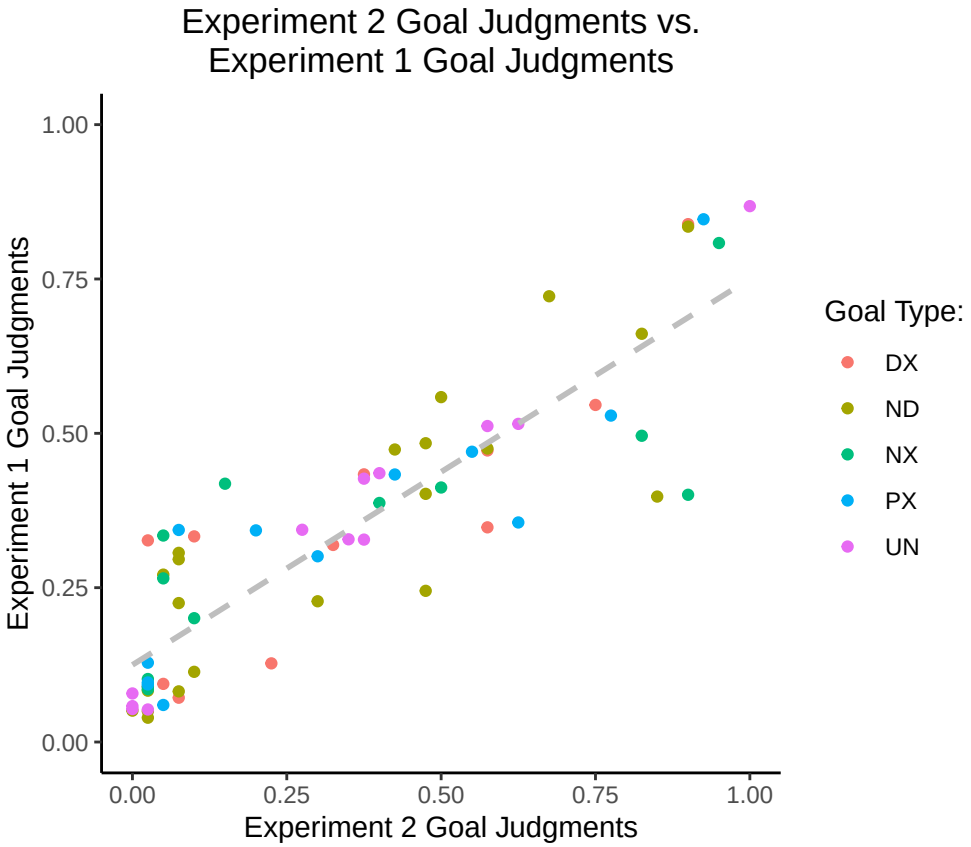
```
    # Stack this participant's data with the rest.
    data_18 = data_18 %>%
      rbind(data_17)
  }
}

# Extract the goals and entrances from the participant-generated paths.
data_20 = tibble()
for (p in 1:length(unique(data_18$participant))) {
  for (m in 1:length(unique(data_18$map))) {
    # Filter the goal coordinates for each participant-generated path and
    # assign the corresponding goal label given its coordinates.
    goals = data_18 %>%
      filter(participant==unique(data_18$participant)[p],
             map==unique(data_18$map)[m]) %>%
      filter(time==max(time)) %>%
      mutate(goal=ifelse(x==2 & y==2, "A",
                         ifelse(x==8 & y==2, "B", "C")))

    # Filter the entrance coordinates for each participant-generated path.
    entrances = data_18 %>%
      filter(participant==unique(data_18$participant)[p],
             map==unique(data_18$map)[m]) %>%
      filter(time==0)

    # Assign the corresponding entrance label given its coordinates.
    for (d in 1:length(doors[[m]])) {
      if (doors[[m]][[d]][1]==entrances$x & doors[[m]][[d]][2]==entrances$y) {
        entrances = entrances %>%
          mutate(entrance=as.character(d))
      }
    }

    # Merge the data back together.
    data_19 = goals %>%
      select(participant, map, goal) %>%
      left_join(select(entrances, participant, map, entrance))

    # Stack the data for this particular participant-generated path.
    data_20 = data_20 %>%
      rbind(data_19)
  }
}
```

Goal Inference

First, we test whether the goals extracted from the participant-generated paths in Experiment 2 predict the goal inferences participants made in Experiment 1.

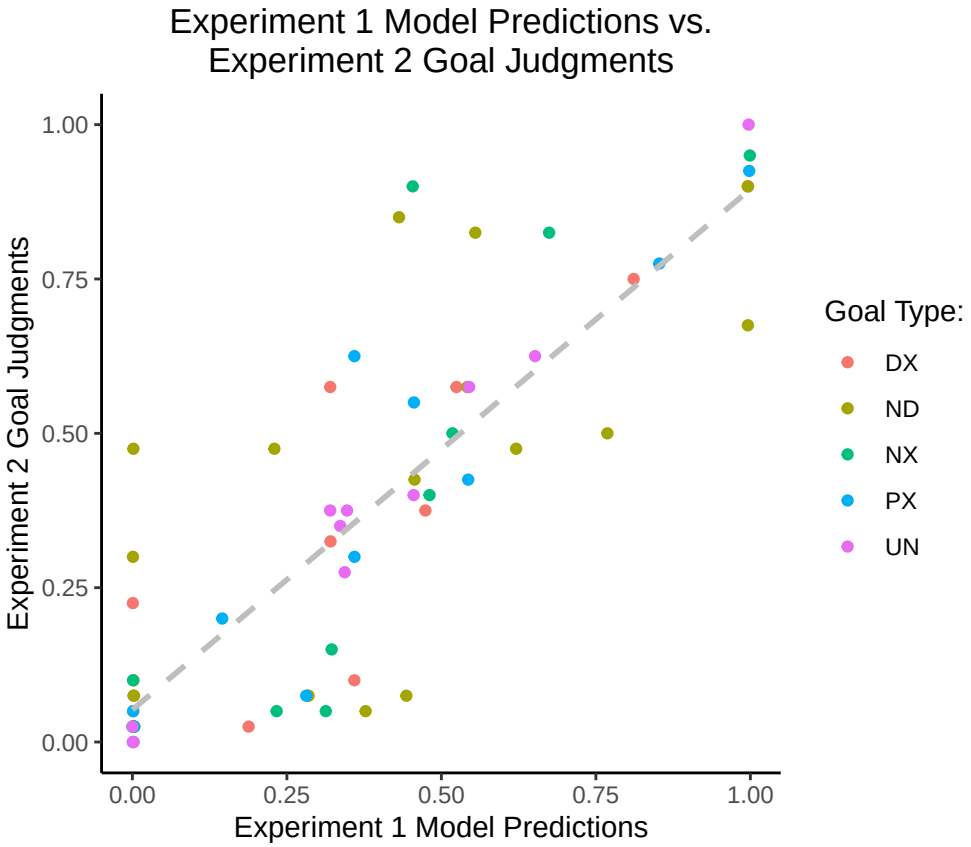
```
# Filter the goal inferences from Experiment 2 and compute the endorsement for
# each goal on each trial.
data_21 = data_20 %>%
```

```
  select(-entrance) %>%
  group_by(map, goal) %>%
  summarize(experiment_2=n()/length(unique(data_20$participant)))

# Merge the goal inferences from Experiment 2 and Experiment 1.
data_22 = data_9 %>%
  select(map, goal, mean_human) %>%
  rename(experiment_1=mean_human) %>%
  left_join(data_21) %>%
  mutate(experiment_2=ifelse(is.na(experiment_2), 0, experiment_2))

# Plot the goal inference comparison.
plot_3 = data_22 %>%
  ggplot(aes(x=experiment_2, y=experiment_1, label=map)) +
  geom_point(aes(color=substr(map, 0, 2))) +
  geom_smooth(method="lm", se=FALSE, linetype="dashed", color="grey") +
  ggtitle("Experiment 2 Goal Judgments vs.\nExperiment 1 Goal Judgments") +
  xlab("Experiment 2 Goal Judgments") +
  ylab("Experiment 1 Goal Judgments") +
  ylim(0.0, 1.0) +
  scale_color_discrete(name="Goal Type:") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        plot.title=element_text(hjust=0.5),
        legend.title=element_text(hjust=0.5))
plot_3
```



The Pearson correlation is r=0.88 (95% CI: 0.8-0.93). By the law of transivity, the goals extracted from these participant-generated paths in Experiment 2 should also correlate with our model's predictions in Experiment 1.

```
# Merge the model predictions from Experiment 1 with the participant goal
# inferences from Experiment 2.
data_23 = data_9 %>%
  select(map, goal, model) %>%
  left_join(data_21) %>%
  mutate(experiment_2=ifelse(is.na(experiment_2), 0, experiment_2))

# Plot the goal inference comparison.
plot_4 = data_23 %>%
  ggplot(aes(x=model, y=experiment_2, label=map)) +
  geom_point(aes(color=substr(map, 0, 2))) +
  geom_smooth(method="lm", se=FALSE, linetype="dashed", color="grey") +
  ggtitle("Experiment 1 Model Predictions vs.\nExperiment 2 Goal Judgments") +
  xlab("Experiment 1 Model Predictions") +
  ylab("Experiment 2 Goal Judgments") +
  ylim(0.0, 1.0) +
  scale_color_discrete(name="Goal Type:") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        plot.title=element_text(hjust=0.5),
        legend.title=element_text(hjust=0.5))
plot_4
```



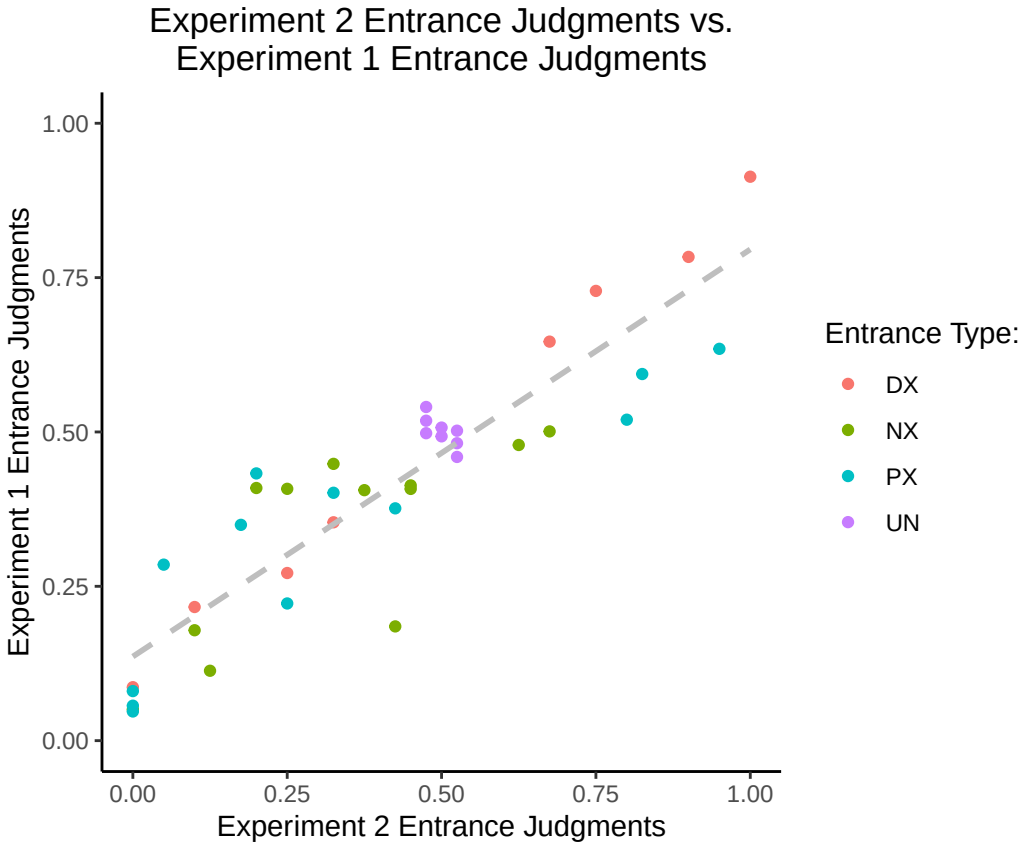
The Pearson correlation is r=0.85 (95% CI: 0.76-0.91).

Entrance Inference

Next, we test whether the entrances extracted from the participant-generated paths in Experiment 2 predict the entrance inferences participants made in Experiment 1.

```
# Filter the entrance inferences from Experiment 2 and compute the endorsement
# for each entrance on each trial.
data_24 = data_20 %>%
  select(-goal) %>%
  group_by(map, entrance) %>%
  summarize(experiment_2=n()/length(unique(data_20$participant)))

# Merge the entrance inferences from Experiment 2 and Experiment 1.
data_25 = data_12 %>%
  select(map, entrance, mean_human) %>%
  rename(experiment_1=mean_human) %>%
  left_join(data_24) %>%
  mutate(experiment_2=ifelse(is.na(experiment_2), 0, experiment_2))

# Plot the entrance inference comparison.
plot_5 = data_25 %>%
  ggplot(aes(x=experiment_2, y=experiment_1, label=map)) +
  geom_point(aes(color=substr(map, 0, 2))) +
```

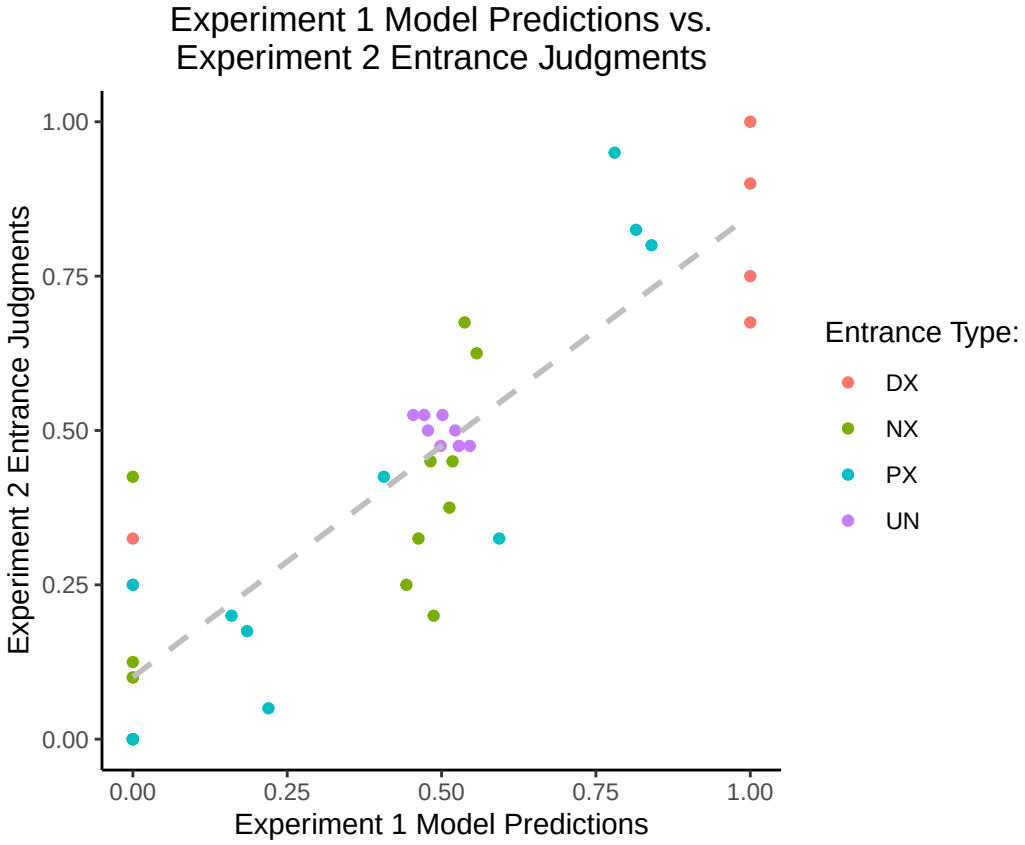
```
  geom_smooth(method="lm", se=FALSE, linetype="dashed", color="grey") +
  ggtitle(paste("Experiment 2 Entrance Judgments vs.\nExperiment 1 Entrance",
                "Judgments")) +
  xlab("Experiment 2 Entrance Judgments") +
  ylab("Experiment 1 Entrance Judgments") +
  ylim(0.0, 1.0) +
  scale_color_discrete(name="Entrance Type:") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        plot.title=element_text(hjust=0.5),
        legend.title=element_text(hjust=0.5))
plot_5
```



The Pearson correlation is r=0.9 (95% CI: 0.82-0.95). By the law of transivity, the entrances extracted from these participant-generated paths in Experiment 2 should also correlate with our model's predictions in Experiment 1.

```
# Merge the model predictions from Experiment 1 with the participant entrance
# inferences from Experiment 2.
data_26 = data_12 %>%
  select(map, entrance, model) %>%
  left_join(data_24) %>%
  mutate(experiment_2=ifelse(is.na(experiment_2), 0, experiment_2))

# Plot the entrance inference comparison.
```

```
plot_6 = data_26 %>%
  ggplot(aes(x=model, y=experiment_2, label=map)) +
  geom_point(aes(color=substr(map, 0, 2))) +
  geom_smooth(method="lm", se=FALSE, linetype="dashed", color="grey") +
  ggtitle(paste("Experiment 1 Model Predictions vs.\nExperiment 2 Entrance",
                "Judgments")) +
  xlab("Experiment 1 Model Predictions") +
  ylab("Experiment 2 Entrance Judgments") +
  ylim(0.0, 1.0) +
  scale_color_discrete(name="Entrance Type:") +
  theme_classic() +
  theme(aspect.ratio=1.0,
        plot.title=element_text(hjust=0.5),
        legend.title=element_text(hjust=0.5))
plot_6
```

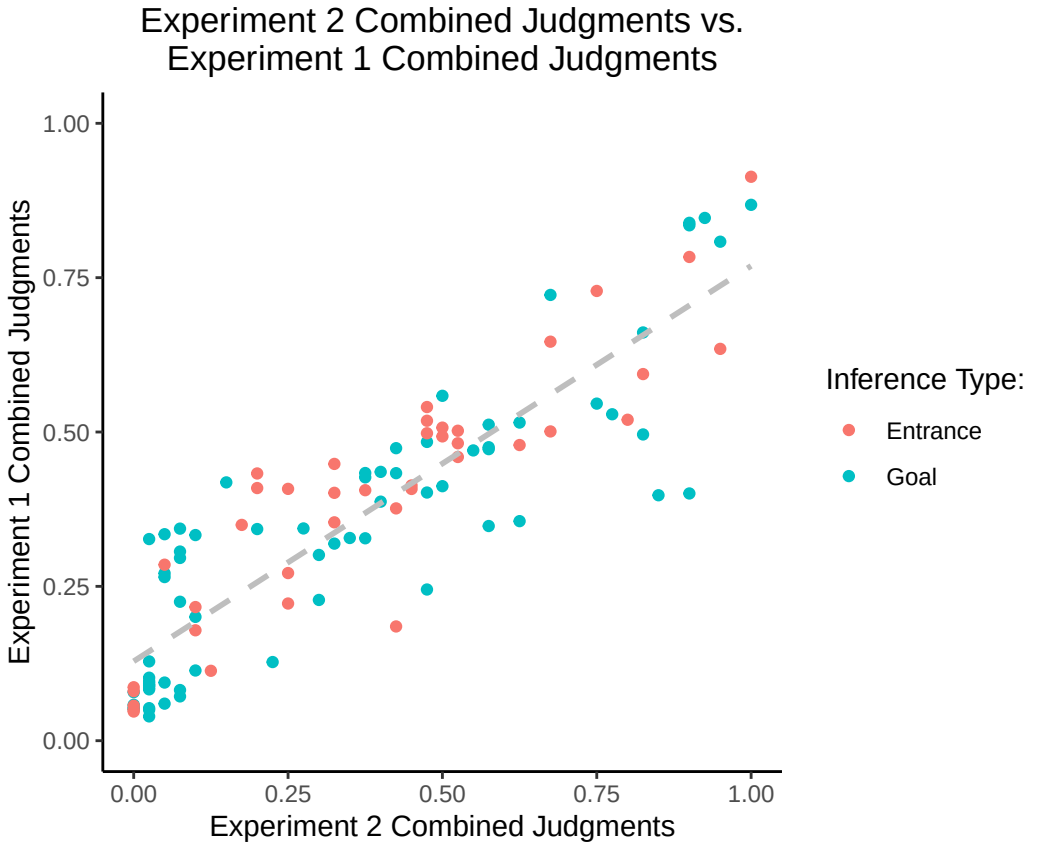


The Pearson correlation is r=0.88 (95% CI: 0.78-0.93).

Combined Inference

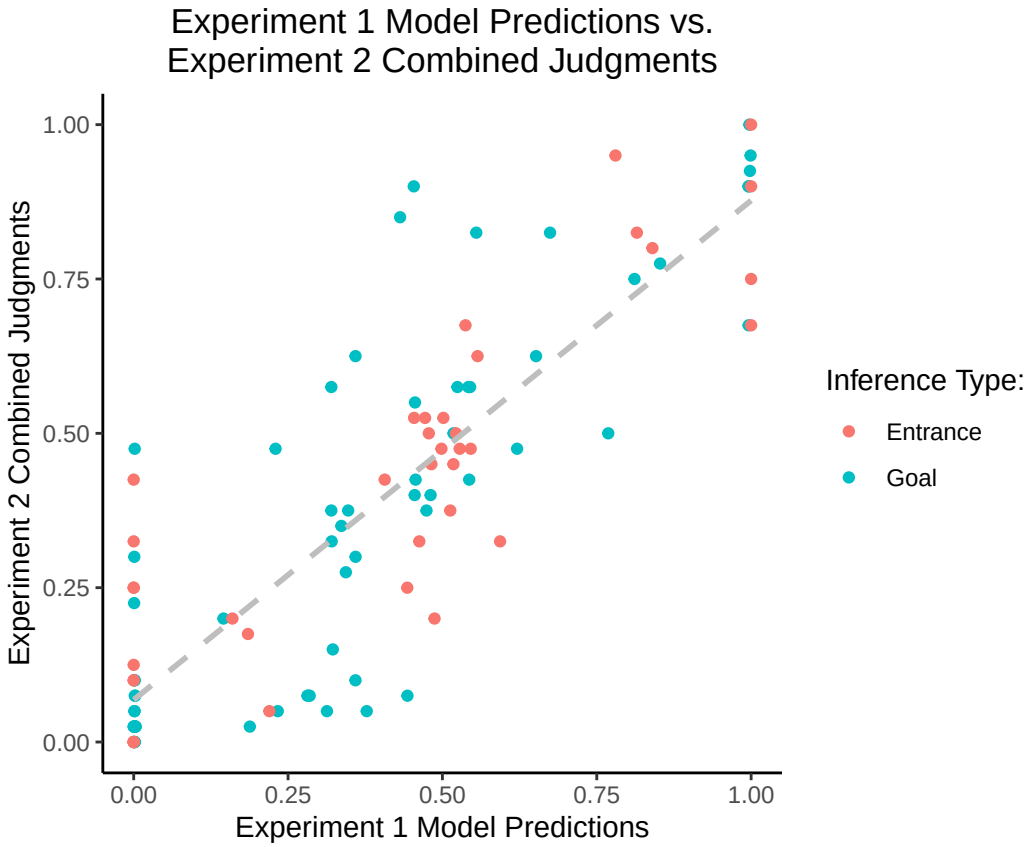
Finally, we jointly analyze the goals and doors extracted from participant-generated paths in Experiment 2 with the goal and entrance inferences participants made in Experiment 1.

```r
# Merge the goal and entrance inferences from Experiment 2 and Experiment 1.
data_27 = data_22 %>%
  mutate(inference=goal, type="goal") %>%
  select(-goal) %>%
  rbind(select(mutate(data_25, inference=entrance, type="entrance"),
               -entrance))

# Plot the combined inference comparison.
plot_7 = data_27 %>%
  ggplot(aes(x=experiment_2, y=experiment_1, label=map)) +
  geom_point(aes(color=type)) +
  geom_smooth(method="lm", se=FALSE, linetype="dashed", color="grey") +
  ggtitle(paste("Experiment 2 Combined Judgments vs.\nExperiment 1 Combined",
                "Judgments")) +
  xlab("Experiment 2 Combined Judgments") +
  ylab("Experiment 1 Combined Judgments") +
  ylim(0.0, 1.0) +
  scale_color_discrete(name="Inference Type:",
                       limits=c("entrance", "goal"),
                       labels=c("Entrance", "Goal")) +
  theme_classic() +
  theme(aspect.ratio=1.0,
        plot.title=element_text(hjust=0.5),
        legend.title=element_text(hjust=0.5))
plot_7
```



The Pearson correlation is r=0.89 (95% CI: 0.83-0.92). Finally, we generate the scatter plot comparing the goals and doors extracted from the participant-generated paths in Experiment 2 against our model's predictions in Experiment 1.

```
# Merge the goal and entrance inferences from Experiment 2 with the model
# predictions from Experiment 1.
data_28 = data_23 %>%
  mutate(inference=goal, type="goal") %>%
  select(-goal) %>%
  rbind(select(mutate(data_26, inference=entrance, type="entrance"),
               -entrance))

# Plot the combined inference comparison.
plot_8 = data_28 %>%
  ggplot(aes(x=model, y=experiment_2, label=map)) +
  geom_point(aes(color=type)) +
  geom_smooth(method="lm", se=FALSE, linetype="dashed", color="grey") +
  ggtitle(paste("Experiment 1 Model Predictions vs.\nExperiment 2",
                "Combined Judgments")) +
  xlab("Experiment 1 Model Predictions") +
  ylab("Experiment 2 Combined Judgments") +
  ylim(0.0, 1.0) +
  scale_color_discrete(name="Inference Type:",
                       limits=c("entrance", "goal"),
                       labels=c("Entrance", "Goal")) +
  theme_classic() +
  theme(aspect.ratio=1.0,
        plot.title=element_text(hjust=0.5),
        legend.title=element_text(hjust=0.5))
plot_8
```



The Pearson correlation is r=0.86 (95% CI: 0.79-0.91).

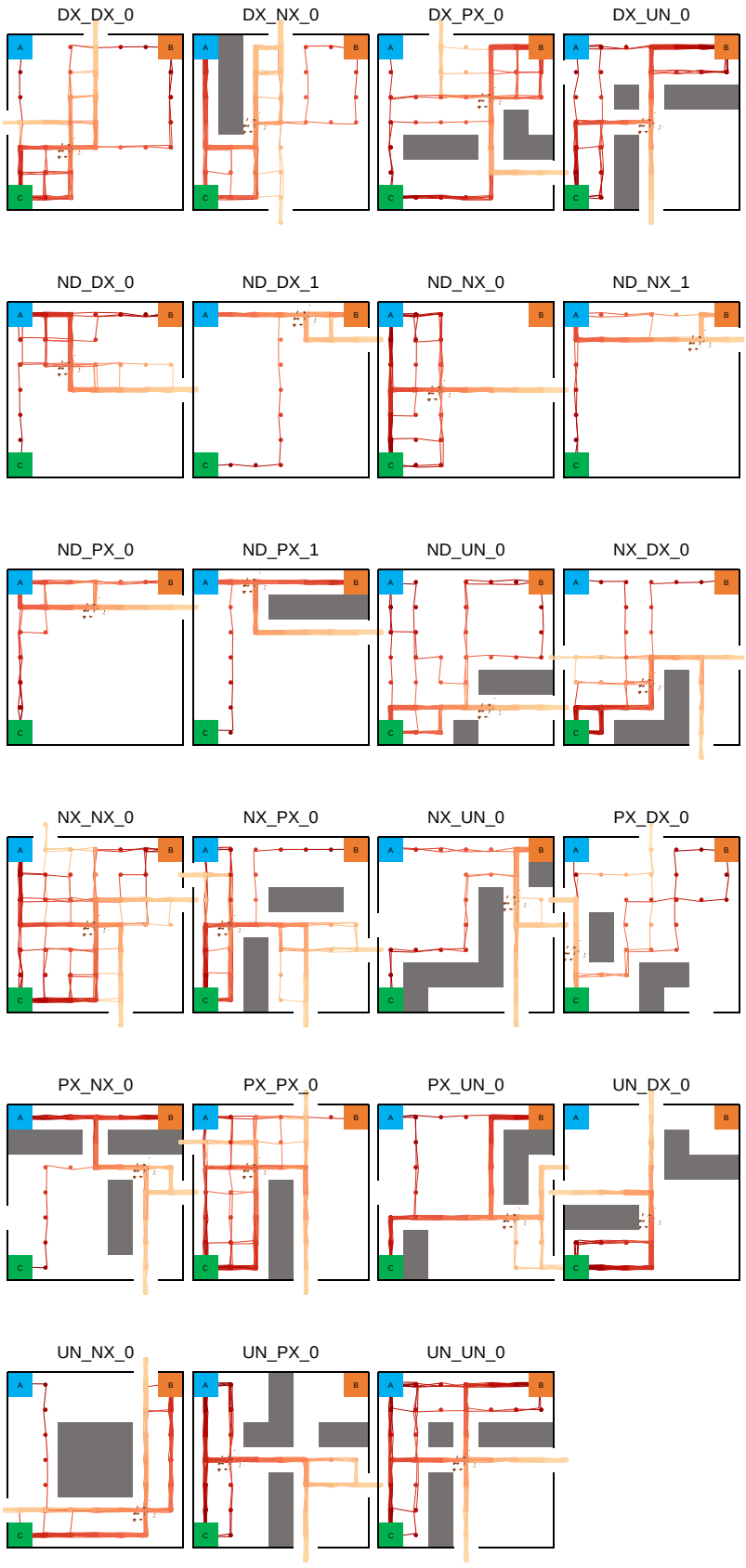
Visualizations

Here we plot the visualizations of the participant-generated data on each trial.

```
# Import the coordinates of the observations, inner walls, and outer walls.
source("stimuli/experiment_2/map_specifications.R")

# Define the coordinates of the goals.
goals = data.frame(
  x_1 = c(2.5, 8.5, 2.5),
  x_2 = c(3.5, 9.5, 3.5),
  y_1 = c(2.5, 2.5, 8.5),
  y_2 = c(3.5, 3.5, 9.5),
  label = c("A", "B", "C")
)

# Define the coordinate bounds.
map_height = 11
map_width = 11

# Set up the general color palette we're going to visualize with.
palette = colorRampPalette(brewer.pal(9, "OrRd"))
```

```
# Generate the visualizations of the participant-generated paths for each
# trial.
plot_10 = list()
plot_11 = list()
for (trial_num in 1:length(unique(data_18$map))) {
  # Extract the coordinates of the observation for this trial.
  crumbs = data.frame(
    x=observations[[trial_num]][1],
    y=observations[[trial_num]][2],
    filename="stimuli/experiment_2/crumbs.png"
  )

  # Compute the specific color palette based on the longest path for this map.
  max_path_length = max(filter(data_18, map==maps[trial_num])$time) + 1
  num_colors = round(max_path_length*1.4)
  color_gradient = palette(num_colors)[(num_colors-max_path_length):num_colors]

  # Generate the visualization for this trial.
  plot_9 = data_18 %>%
    filter(map==maps[trial_num]) %>%
    mutate(x=x+1, y=y+1) %>%
    ggplot() +
    geom_path(aes(x=x, y=y, group=participant, color=time),
              position=position_jitter(height=0.1, width=0.1)) +
    geom_point(aes(x=x, y=y, group=participant, color=time)) +
    scale_x_continuous(name="", limits=c(0, map_width+1)) +
    scale_y_reverse(name="", limits=c(map_height+1, 1)) +
    theme_void() +
    theme(legend.position="none") +
    coord_fixed() +
    geom_image(data=crumbs, aes(x=x, y=y, image=filename), size=0.1) +
    geom_rect(data=goals,
              aes(xmin=x_1, xmax=x_2, ymin=y_1, ymax=y_2, fill=label),
              alpha=1.0) +
    geom_text(data=goals, aes(label=label, x=(x_1+x_2)/2, y=(y_1+y_2)/2),
              size=9*(5/14)) +
    scale_color_gradientn(colors=color_gradient,
                          limits=c(0, max_path_length-1)) +
    scale_fill_manual(values=c("#00AfEf", "#ED7D31", "#00B050", "#767171"))

  # Add the walls to the visualization.
  for (wall in walls[trial_num][[1]]) {
    plot_9 = plot_9 + wall
  }

  # Add the border to the visualization.
  for (segment in segments[trial_num][[1]]) {
    plot_9 = plot_9 + segment
  }

  # Store the plot separately before adding a title and adjusting the margins.
  plot_10[[trial_num]] = plot_9
```

```
  # Add a title to the visualization, adjust the margins, and store it.
  plot_11[[trial_num]] = plot_9 +
    ggtitle(maps[trial_num]) +
    theme(plot.title=element_text(size=20, hjust=0.5, vjust=-10.0),
          plot.margin=unit(c(-2, -3, -2, -3), "cm"))
}

# Plot the visualizations in a grid.
plot_12 = plot_grid(plot_11[[1]], plot_11[[2]], plot_11[[3]], plot_11[[4]],
                    plot_11[[5]], plot_11[[6]], plot_11[[7]], plot_11[[8]],
                    plot_11[[9]], plot_11[[10]], plot_11[[11]], plot_11[[12]],
                    plot_11[[13]], plot_11[[14]], plot_11[[15]], plot_11[[16]],
                    plot_11[[17]], plot_11[[18]], plot_11[[19]], plot_11[[20]],
                    plot_11[[21]], plot_11[[22]], plot_11[[23]],
                    ncol=4)
plot_12
```


DX_DX_0
DX_NX_0
DX_PX_0
DX_UN_0
ND_DX_0
ND_DX_1
ND_NX_0
ND_NX_1
ND_PX_0
ND_PX_1
ND_UN_0
NX_DX_0
NX_NX_0
NX_PX_0
NX_UN_0
PX_DX_0
PX_NX_0
PX_PX_0
PX_UN_0
UN_DX_0
UN_NX_0
UN_PX_0
UN_UN_0

Saving the visualizations

Here we save the visualizations (excluding the title and margin adjustments).

```
# Save the visualizations.
for (trial_num in 1:length(unique(data_18$map))) {
  # Stitch the path for this visualization.
  data_path = paste("analysis/experiment_2/visualizations/",
                    maps[trial_num], sep="")

  # Save this visualization.
  ggsave(paste(data_path, ".png", sep=""), plot_10[[trial_num]], device="png",
         height=4.5, width=7.3)
  ggsave(paste(data_path, ".pdf", sep=""), plot_10[[trial_num]], device="pdf",
         height=4.5, width=7.3)
}
```